14 PASKAITA

Turinys: Šablonai
Funkcijų šablonai. Klasių šablonai. Įvadas į STL biblioteką.

Funkcijų šablonai

Šablonas (template) – mechanizmas, leidžiantis trumpai užrašyti identiškų, tik argumentais besiskiriančių funkcijų seką. Pavyzdžiui, reikia tokių perkrautų modulio funkcijų įvairiems duomenų tipams:

```
int abs( int n ){
        return ( n<0 ) ? –n : n;
}
//
float abs( float n ){
        return ( n<0 ) ? –n : n;
}
//
. . .
```

Visas tokias funkcijas galima užrašyti viena šablonine funkcija:

```
template <class T>    // arba <typename T>
T abs( T n ){
        return( n<0 ) ? –n : n;
}
```

Šablone teikiamas apibendrintas duomens tipas – *T*, vadinamas šablono argumentu. Vėliau, kompiliatoriui kviečiant šabloninę funkciją su konkrečiu vidinio tipo argumentu, kompiliatorius sukuria funkciją, operuojančią to tipo argumentu. Jei funkcijoje esantys veiksmai apibrėžti (t.y. veiksmų operatoriai atitinkamai perkrauti) vartotojo duomens tipui, šabloninei funkcijai galima teikti ir tokį argumentą.
Pavyzdys su ta pačia funkcija skirtingiems duomenų tipams:

```
#include <iostream>
using namespace std;
//
template <class T>
T abs( T n ){
        return( n<0 ) ? –n : n;
}
//
int main( ){
        int i = -10;
        long li = -10;
        float f = -10.f;
                cout<<abs( i )<<endl;
                cout<<abs( li )<<endl;
```

```
        cout<<abs( f )<<endl;
                return 0;
}
```

Kompiliatoriaus darbas su šablonais: funkcijos kodas generuojamas tik vykdymo metu iškvietus funkciją su konkrečiu tipu. Tai – „funkcijos šablono realizavimas“. Jei yra keli kreipiniai su tokiu tipu, generuojamas tik vienas funkcijos kodas.
Pavyzdys: funkcija su keliais vieno tipo šablono argumentais. Masyve ieškoma duotosios reikšmės. Jei tokia masyve yra, grąžinamas tos reikšmės pirmosios aptikties masyve indeksas, o jei nėra – -1.

```
#include <iostream>
using namespace std;
//
template <class t>
int find( t* array, t value, int size ){
        for( int i = 0; i<size; i++ )
                if( array[ i ] == value ) return i;
        return –1;
}
//
int main( ){
        int iarray[ ]       = { 1, 2, 3, 4, 5 };
        double darray[ ] = { 1., 2., 3., 4., 5. );
        int ivalue = 3;
        double dvalue = 3.;
                cout<<find( iarray, ivalue, 5 )<<endl;
                cout<<find( darray, dvalue, 5 )<<endl;
                        return 0;
}
```

Pavyzdys: keli skirtingo tipo šablono argumentai. Uždavinys – tas pat, tik tarkim, masyvas gali būti ir labai didelio matmens – tokiu atveju *size* turėtų būti *long* tipo.

```
#include <iostream>
using namespace std;
//
template <class tarray, class tsize >
tsize find( tarray* array, tarray value, tsize size ){
        for( tsize i = 0; i<size; i++ )
                if( array[ i ] == value ) return i;
        return static_cast< tsize >( –1  );
}
//
int main( ){
        int iarray[ ]       = { 1, 2, 3, 4, 5 };
        double darray[ ] = { 1., 2., 3., 4., 5. );
        int ivalue = 3;
        double dvalue = 3.;
                cout<<find( iarray, ivalue, 5 )<<endl;
```

```
        cout<<find( darray, dvalue, 5 )<<endl;
            return 0;
}
```

Klasių šablonai

Labai panašūs į funkcijų šablonus.
Pavyzdys: jau matyta klasė *Stack*, perdirbta taip, kad stekas galėtų saugoti visų tipų duomenis.

```
#include <iostream>
using namespace std;
//
const int MAX = 100;  // steko dydis
//
template <class t>
class Stack{
    private:
        t st[ MAX ];
        int top;
    public:
        Stack( ){  top = -1; }
        void push( t value ){
            st[++top ] = value;
        }
        t pop( ){
            return st[ top-- ];
        }
};
//
int main( ){
    Stack < float > s1;  // būtina nurodyti šablono argumento tipą
    Stack < int > s2;
        s1.push( 1.f );
        s1.push( 2.f );
        cout<<"1: "<<s1.pop( )<<endl;
        cout<<"1: "<<s1.pop( )<<endl;
            s2.push( 1 );
            s2.push( 2 );
            cout<<"2: "<<s1.pop( )<<endl;
            cout<<"2: "<<s1.pop( )<<endl;
                return 0;
}
```

Klasės-šablono metodus galima nurodyti ir už klasės ribų:

```
. . .
template <class t>
class Stack{
```

```
        private:
                t st[ MAX ];
                int top;
        public:
                Stack( );   // prototipai
                void push( t );
                t pop( );
};
//
template < class t >   // būtinas raktažodis template
Stack < t > :: Stack( ){  // būtinas argumento tipas ir priklausomybės operacija
        top = -1;
}
//
template < class t >
void Stack < t > ::push( t value ){
        st[ ++top ] = value;
}
//
template < class t >
t Stack < t > :: pop( ){
        return st[ top-- ];
}
. . .
```

Įvadas į STL

Tai – duomenų struktūrų ir algoritmų jiems apdoroti biblioteka. Sukūrė A.Stepanov ir M.Li iš HP; dabar įjungta į ISO/ANSI C++ standartą. Visų STL objektų vardai įtraukti į vardų sritį *std*.
Biblioteką sudaro trys dalys:

- duomenų struktūros („konteineriai“). Realizuoti šabloninėmis klasėmis, kad galima būtų talpinti įvairių tipų duomenis;
- algoritmai konteineriuose esančių duomenų apdorojimui – nepriklausomos nuo klasių šabloninės funkcijos;
- iteratoriai – objektai, leidžiantys perrinkti konteinerio elementus.

Konteineriai: nuoseklieji (masyvai, vektoriai, sąrašai, eilės) ir asocijuotieji (aibės, atvaizdžiai, hash-struktūros).
Algoritmai: paieškos, kopijavimo, sukeitimo, apjungimo, rūšiavimo ir pan.
Iteratoriai: objektai, leidžiantys judėti konteineryje viena ar abiem kryptimis, prijungia konteinerius prie įvesties/išvesties srautų. „Gaunami“ iš pačių konteinerių.

Pagrindiniai konteineriai:

vector<T> (kad būtų galima naudoti šį konteinerį, reikia antraštinio failo <*vector*>) – vienmatis dinaminis masyvas bet kokiems vidiniams ar vartotojo sukurto duomenų tipo objektams saugoti. Masyvas automatiškai plečiasi įdedant į jį naujus duomenis (tačiau duomenis galima pridėti tik masyvo gale). Programuotojui rūpintis atminties masyvui skyrimu ir atlaisvinimu nereikia.

deque<T> (antraštinis failas <*deque*>) – abipusė eilė – panaši į *vector* duomenų struktūra, tik duomenis galima įdėti ir iš priekio, ir iš galo.

list<T> (<*list*>) – dvipusis sąrašas.

map<K, T> (<*map*>) – paprasčiausias asociatyvinis konteineris, kriame duomenys saugomi poromis raktas/reikšmė; pagal raktą galima išrinkti atitinkamą reikšmę. Duomenų bazės prototipas.

set<T> (<*set*>) – aibė – panaši į struktūra į map, kuriame būtų saugomos tik reikšmės. Leidžiamos tik unikalios reikšmės.

bitset<T> (<*bitset*>) – bitų aibė; dažniausiai šioje struktūroje saugomos vėliavėlių reikšmės.

Pagrindinės iteratorių rūšys:

įvesties/išvesties – gali būti panaudoti programoje tik po vieną sykį;
į priekį – judėjimui konteineryje į priekį;
dvikrypčiai;
atsitiktinės kreipties – pagal nurodytą objekto konteineryje numerį (indeksą).

Algoritmai (kad juos būtų galima naudoti, reikia antraštinio failo <*algorithm*>):

find – surasti objektą konteineryje;
replace – pakeisti objektą (arba jų grupę) kitu (kita grupe);
fill – užpildyti visą (dalį) konteinerio duotais objektais;
transform – pakeisti visą (dalį) konteinerio objektų nurodytu būdu – vadinamuoju lyginimo predikatu (žr. žemiau);
sort – rūšiuoti visą (dalį) konteinerio nurodyta tvarka, taip pat tam būtini predikatai;
. . .

Predikatai (jiems reikia antraštinio failo <*functional*>):

less<T>
less_equal<T>
equal<T>
greater_equal<T>
greater<T>

Pavyzdys: programa-schema darbui su *vector* objektais.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>
using namespace std;
//
int main(){
        vector<int> v1(100);  //vector objektas iš 100 elementų; visi
                              // inicializuoti nulinėmis reikšmėmis
        vector<double> v2(10,-1.); //inicializuojama reikšmėmis -1.
        . . .
        v1[0] = 50;     // pirmajam elementui suteikti reikšmę 50
        v2.push_back(1.);   //11-ajam elementui priskirti 1.
        . . .
```

```
v1.resize(200);  // v1 paskirti dvigubai atminties
. . .
//prieiga prie elementų
for( vector<int>::size_type i=0; i<v1.size(); i++)
        cout<<v1.at(i);
. . .
//rūšiavimas mažėjimo tvarka
sort( v1.begin(), v1.end(), greater<int>() );
. . .
```