C++ programavimo kalba

Šablonai

(10 paskaita)

Kodėl šablonai (templates)?

Programuojant egzistuoja situacijos, kai reikia atlikti tuos pačius veiksmus su skirtingais duomenų tipais (pvz. modulio radimas, kėlimas laipsniu).
Tam patogu naudoti **šablonines** arba **parametrizuotas** funkcijas ir klases.

- Šablonai leidžia sukurti bendro pobūdžio funkcijas ir klases, neprisirišant prie konkrečių duomenų tipų.
- Leidžia kurti bendro tipo bibliotekas (STL) ir taip užtikrina daugkartinį funkcijų ar klasių panaudojimą.
- Šablonų tipai : funkcijų šablonai, klasių šalonai.

Pavyzdys:

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

C kalboje egzistuoja abs(); fabs(); fabsl(); labs(), cabs()...

Funkcijų šablonai (function template)

Funkcijos šablonas visad apibrėžiamas raktiniu žodžiu **template** ir vienu ar keliais parametrais .

```
template <class T> Max (T, T);

template <class T> T abs(T n)
{return (n<0)? –n, n;}
```

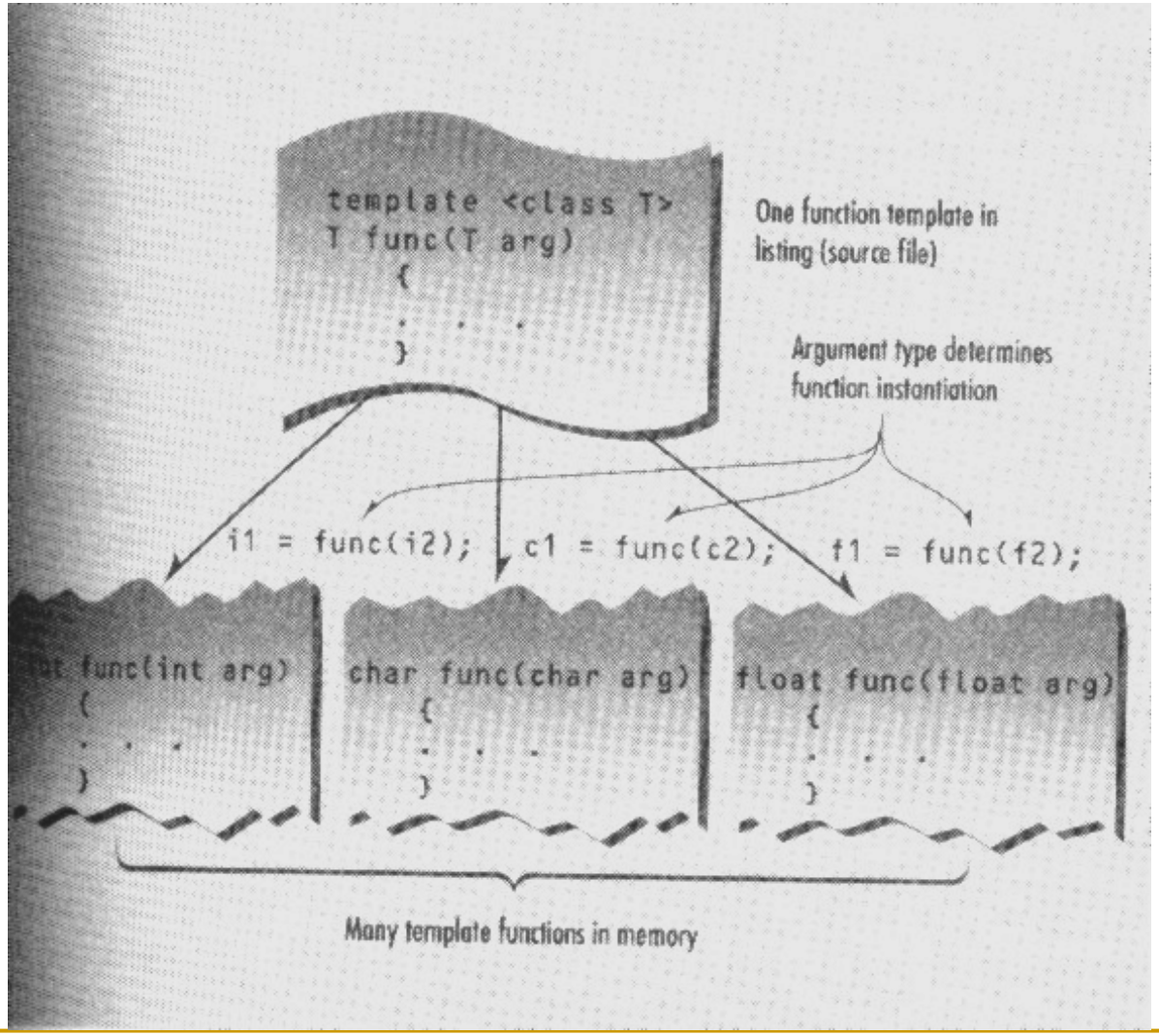
```
template <class A> A Sqr (A skc)
{ return skc * skc;  }
```

template <*typename* identifier> function_declaration
template <*class* identifier> function_declaration → Sintaksė

Argumentai, esantys <> viduje vadinami šablono argumentais.
class – raktinis žodis. Jis būtinas prieš kiekvieną argumentą!!!

Kompiliatorius kuria konkrečią funkcijos realizaciją (šabloninę funkciją – template function) programos vykdymo metu.
Kaip atrodys konkreti funkcija, priklauso nuo argumentų tipo.

Funkcijų šablonai



Funkcijų šablonai su keletu argumentų

```
template <class T1, class T2, class T3>
        T3 Relation(T1, T2);                    // ok
template <class T1, class T2>
        int Compare (T1, T1);                   // blogai! T2 neneaudojamas.
template <class T1, T2>                         // blogai! class praleistas dėl T2
        int Compare (T1, T2);
```

Pavyzdys

```
template <class T>
T Max (T val1, T val2)
{
        return val1 > val2 ? val1 : val2;
}
cout << Max(19, 5) << ' ' << Max(10.5, 20.3) << ' ' << Max('a','b') << '\n'
```

Pavyzdys

```
#include <iostream>
using namespace std;

template <class atype, class btype>
int find(atype* array, atype value, btype size)
{ for (btype j=0; j<size; j++)
    if (array[ j ] == value)
        return j;
    return -1; }

void main( )
{int size = 5;
 char chArr [ ] = {1, 2, 3, 4, 5};      char ch = 5;
 int intArr [ ] = {1, 2, 3, 4, 5};      int in = 4;
 double douArr[ ] = {1.0, 2.0, 3.0, 4.0, 5.0 };
 double db = 4.0;

 cout << “char” << find (chArr, ch, size)<<endl;
 cout << “int ” << find (intArr, in, size)<<endl;
 cout << “char ” << find (douArr, db, size)<<endl; }
```

Šabloninių funkcijų perkrovimas

Kaip ir paprastos taip ir šabloninės funkcijos gali būti **perkrautos** t.y. egzistuoti keletas šabloninių funkcijų tais pačiais pavadinimais, bet skirtingais argumentų sąrašais.

```
template <class T>
const T max( const T a, const T b)
{ return a >b ?  a :b ;  }
```

```
template <class T>
const T max( T* a, int size)
{ T* tmp = a;
   for (int i = 0; i < size; i++ )
    { if( a[i] > *tmp )
      *tmp = a[i];
     }
   return *tmp;
}
```

```
void main ()
{int m = 9, n = 12;
 int arr[ ] = { 3, 4, 5, 7, 9 };
 cout << "Max int = " << max(m, n)<< endl;
 cout << "Max arr = ";
 cout  << max(arr, sizeof(arr)/sizeof(int) )<<
              endl;
}
```

Šablonų ir macros skirtumai

Macros

```
# define abs(n) ( (n<0) ? (-n) : (n) )
```

Problemos:

1. MACROS netikrina argumentų duomenų tipo;
2. MACROS netikrina gražinamos reikšmės tipo.

Klasių šablonai

Šablonų koncepsija gali būti išplėsta pritaikant ją klasėms. Šablonai naudojami specialioms klasėms, kurios vadinamos duomenų (container) klasėms (pvz. stekams (stacks) ir sujungtiems sąrašams (linked lists)).

Sintaksė:

```
template <class Tipas> class klasės_pavadinimas
{  };
```

```
template <class Type>
class Stack
 { private: Type st[10];
            int top;
  public: Stack() {top = -1;}
  void push (Type var) { st[++top] = var; }
  Type pop () { return st [top--]; }
 };
```

```
void main()
{Stack<float> s1;
  s1.push(11.1F);
  cout<<s1.pop()<<endl;
  Stack<long> s2;
  s2.push(123L);
  cout<<s2.pop()<<endl;
}
```

Klasių šablonų konkretizacija (Class Template Instantiation)

Klasės sukūrimo procesas pagal turimą šabloną vadinamas klasės šablono konkretizacija (Instantiation). Šabloninė klasė konkretizuojama, prijungiant prie jos vardo pilną faktinių argumentų sąrašą programos kompiliavimo metu. Tada kiekvienas parametras <class ...> keičiamas konkrečiu duomenų tipu.

```
Stack<int>       s1(10);        // stack of integers
Stack<double> s2(10);           // stack of doubles
Stack<float>    s3(10);         // stack of floats
```

Klasių šablonai

```
#include <iostream>
using namespace std;

template <class T>
class pair {
    T a, b;
  public:
    pair (T first, T second)
      {a=first; b=second;}
    T getmax ();
};

template <class T>
T pair<T>::getmax ()
{
  T retval;
  retval = a>b? a : b;
  return retval; }
```

```
int main () {
  pair <int> myobject (100, 75);
  cout << myobject.getmax();
  return 0;
}
```

Klasių šablonai

```
template <class Type>
class Stack
{ private:
        Type *stack;            // steko masyvas
        int top;                // steko indeksas
        const int maxSize;      // maksimalus steko dydis
  public:
    Stack(int max) : stack(new Type[max]), top(-1), maxSize(max)
    { }
   ~Stack(void)
     { delete [ ] stack; }

    void Push (Type &val);
    void Pop (void)   {  if (top >= 0) --top;}
    Type& Top (void) {return stack[top];}
    friend ostream& operator << (ostream&, Stack&);
};
```

Klasių šablonai

Jei klasės funkcija apibrėžiama už klasės ribų naudojama sekanti sintaksė:

```
template <class Type>
void Stack<Type>::Push (Type &val)
{
        if (top+1 < maxSize)
                stack[++top] = val;
}

template <class Type>
ostream& operator << (ostream& os, Stack<Type>& s)
{
        for (int i = 0; i <= s.top; ++i)
                os << s.stack[i] << " ";
        return os;
}
```

Klasių šablonai

```
#include <iostream.h>
using namespace std;
template <class Type>
class Stack
 { private: Type st [10];
            int top;
   public:  Stack() { top = -1; }                    // Konstruktorius
            void push (Type var) { st [ ++top ] = var; }
            Type pop () { return st [top -- ]; }
 };

   void main()
   {  Stack <float> s1;                    // s1 – klasės Stack<float> objektas
      s1.push(11.1F);  s1.push(22.2F);
      cout<<s1.pop()<<endl;
      cout<<s1.pop()<<endl;
      Stack <long> s2;
      s2.push(123L);      cout<<s2.pop()<<endl;    }
```

Non-type Parametrai klasių šablonuose

Klasių šablonai gali turėti ne vien tik neapibrėžto duomenų tipo kintamuosius. Ten gali būti ir kintamieji, kurių tipas iš karto apibrėžiamas.

```
template <class Type, int maxSize>
class Stack {
public:  Stack (void) : stack(new Type[maxSize]), top(-1) { }
         ~Stack (void)  { delete [ ] stack; }
             void Push (Type &val);
             void Pop (void) { if (top >= 0) --top; }
             Type &Top (void)   { return stack [top]; }
private:
             Type *stack;                  // steko masyvas
             int top;                      // steko viršutinio elemento indeksas
};
```

```
             Stack<int, 10> s1;            // ok
             Stack<int, 10,55> s2;         // blogai! 10.55 – ne int tipo
             Stack<int, 5+5> s3;           // ok
             int n = 10;
             Stack<int, n> s4;             // blogai! n  - kintamasis
```

Non-type Parametrai klasių šablonuose

```
#include <iostream>
using namespace std;

template <class T, int N>
class sequence {
    T memblock [N];
  public:
    void setmember (int x, T value);
    T getmember (int x);
};
template <class T, int N>
void sequence<T,N>::setmember (int x, T value) {
  memblock[x]=value;
}
template <class T, int N>
T sequence<T,N>::getmember (int x) {
  return memblock[x];
}
```

```
int main () {
  sequence <int,5> myints;
  sequence <double,5> myfloats;
  myints.setmember (0,100);
  myfloats.setmember (3,3.1416);
  cout << myints.getmember(0) <<
'\n';
  cout << myfloats.getmember(3)
<< '\n';
  return 0;
}
```

Klasių šablonų nariai (Class Template Members)

Klasių šablonų nariai gali būti konstantos, nuorodos, statiniai nariai t.y. tokie pat kintamieji, kaip ir įprastinėse klasėse. Jei šablone yra statiniai kintamieji, kiekviena klasė, sukurta pagal tokį šabloną turi savo statinius kintamuosius.

```
template <class Type>
class Stack {
public:   //...
          Type& Top(void);
private:
          //...
          static Type dummy;          };

template <class Type>
Type& Stack<Type>::Top (void)
{  return top >= 0 ? Stack [top] : dummy; }

template <class Type> Type Stack<Type>::dummy = 0;
```

Klasių šablonai ir draugiškos klasės(funkcijos)

Klasių šablonuose gali būti draugiškos klasės ir funkcijos. Sakykime, kad turime draugišką funkcijos šabloną *Foo* ir draugišką klasės šabloną *Stack*.

```
template <class T>
class Sample {                          // one-to-many friendship
        friend Foo<int>;
        friend Stack<int>;
};
template <class T>
class Sample {                          // one-to-one friendship
        friend Foo<T>;
        friend Stack<T>;
};
template <class T>
class Sample {                          // many-to-many friendship
   template <class X> friend Foo();
   template <class X> friend class Stack();
};
```

Klasių šablonai ir paveldimumas

Klasių šablonai gali būti tiek bazinėmis klasėmis tiek ir išvestinėmis klasėmis.

```
template <class Type>
class SmartList : public List<Type>;          // bazinė klasė šablonas

class MyClass X;
template <class Type> class Y : X;            // išvestinė klasė šablonas

template <class Type>
class Set : public List<Type>
{
   public:  virtual void Insert (const Type &val)
                { if (!Member(val)) List<Type>::Insert(val); }
};
```

Šablonų apibendrinimas

- **Šablonai – tai C++ savybė, leidžianti kurti parametrizuoto tipo funkcijas ir klases, kurių pagalba galima kurti klases ar funkcijas keičiančias savo elgseną priklausomai nuo parametrų**. Šablonai – tai galimybė pakartotinai naudoti jau sukurtus programinius kodus.
- Šablonus apibrėžia parametrizuoti kintamieji.
- Klasių šablonai gali turėti draugiškas klases ir funkcijas bei klasių ir funkcijų šablonus.
- Šablonai gali turėti ir statinius kintamuosius. Tuo atveju kiekviena šablono konkretizacija kuria atskirus savo statinių duomenų rinkinius.