Şablonlar (Templates)

Dr. Metin Özkan

Giriş

- Programcı bazen, aynı fonksiyon ve sınıfın birden çok sürümünü farklı değişken tipleri için yazmak zorunda kalır.

- C++, programcının şablonlar tanımlamasına imkan sağlar. Böylece, fonksiyon ve sınıfların parametrik yazılması mümkün olur :
  - *Bir fonksiyonda, değişken tipleri yerine parametre kullanır.*
  - *Bir sınıfta, veri üyelerinin tipleri parametre olarak tanımlanır.*

Giriş

- Şablonlar (templates), güçlü programlama araçlarıdır
  - *Şablonlar bir kere kodlanır. Kullanılan parametre tipine göre fonksiyonlar ve sınıflar otomatik oluşturulur.*
  - *Aynı şeyleri tekrar kodlamaktan kaynaklanacak hataların önüne geçilmiş olur.*

Fonksiyon Şablonları (Function Templates)

- Integer tip bir sayının mutlak değerini bulmak için bir abs fonksiyonu olsun.

```
int abs (int n){
        if(n<0) return -n;
        else return n;
}
```

- Ayrıca, aynı fonksiyon long tipi sayı içinde istenmektedir.

```
long abs (long n){
        if(n<0) return -n;
        else return n;
}
```

- Hatta, aynı fonksiyon double tipi sayı içinde istenmektedir.

```
double abs (double n){
        if(n<0) return -n;
        else return n;
}
```

Fonksiyon Şablonları (Function Templates)

- Bunun yerine bütün tipler için kullanılabilecek tek bir fonksiyon yazılabilir.

```
template <typename T>
T abs (T n){
        if(n<0) return -n;
        else return n;
}
int main(){
        int a=-2;
        long b=-1000;
        double c=-0.12;
        cout<<abs(a)<<endl; //int abs(int)
        cout<<abs(b)<<endl; //long abs(long)
        cout<<abs(c)<<endl; //double abs(double)
}
```

Fonksiyon Şablonları (Function Templates)

- Fonksiyon şablonları
  - *Farklı veri tipleri için özdeş işlem yürüten yüklenmiş (overloaded) fonksiyonlar yazılmasında kullanılır.*
    - *Programcı, fonksiyon şablon tanımı yazar.*
    - *Derleyici çağrıdaki parametre tipine göre ayrı object kod fonksiyonlar üretir.*
  - *C'deki makrolara benzerdir. Ama, burada tip kontrolü vardır.*
- Fonksiyon şablon tanımaları
  - *Örnekler*
    - `template< typename T >`
    - `template< class ElementType >`
    - `template< typename BorderType, typename Filltype >`

Fonksiyon Şablonları (Function Templates)

- Bir fonksiyon şablonun parametresi nesne olabilir.

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

class A{
          int a;
public:
          A(int q=0){
                    a=q;
          }
          bool operator==(const A &right) const{
                    if (a==right.a)
                              return true;
                    else
                              return false;
          }
};

template <typename T>
bool compare (const T &x, const T &y){
          return x==y;
}

int main(){
          A objA(1), objB(2);
          int a=2, b=2;
          compare(objA, objB);
          compare(a,b);
}
```

Fonksiyon Şablonları (Function Templates)-Bir Örnek

```
// Using template functions.
#include <iostream>
using namespace std;

// function template printArray definition
template< typename T >
void printArray(const T * const array,int count )
{
   for ( int i = 0; i < count; i++ )
      cout << array[ i ] << " ";

   cout << endl;
} // end function template printArray3
```

- printArrray fonksiyonu şablonu, farklı tiplerdeki dizilerdeki (arrays) değerleri belli bir formatta ekrana yazdırır.

```
int main()
{
   const int aCount = 5; // size of array a
   const int bCount = 7; // size of array b
   const int cCount = 6; // size of array c

   int a[ aCount ] = { 1, 2, 3, 4, 5 };
   double b[ bCount ]
           = { 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7 };
   char c[ cCount ] = "HELLO"; // 6th position for null

   cout << "Array a contains:" << endl;

   // call integer function-template specialization
   printArray( a, aCount );

   cout << "Array b contains:" << endl;

   // call double function-template specialization
   printArray( b, bCount );

   cout << "Array c contains:" << endl;

   // call character function-template specialization
   printArray( c, cCount );
} // end main
```

Fonksiyon Şablonları (Function Templates)

- Bir fonksiyon şablonu yüklenebilir (may be overloaded). Yükleme fonksiyonları:
  - *Diğer fonksiyon şablonlar aynı isimde ama farklı parametreye sahip olmalıdır.*
  - *Şablon olmayan fonksiyonlar aynı isimde ama farklı parametrelere sahip olmalıdır.*
- Çağrı gerçekleştirilen fonksiyona en fazla eşleşen fonksiyon derleyici tarafında seçilir. Bu seçim sürecinde,
  - *Şablon olmayan fonksiyon, şablon olana göre öncelikli seçilir.*
  - *Aksi halde, birden çok eşleşme derleme hatasına neden olur.*

Sınıf Şablonları (Class Templates)

- Sınıflarda şablon olarak oluşturulabilir.
- Sınıf şablonlar, genellikle veri depolama (container) kullanılır.

- Sınıf şablonlar,
  - *Sınıf şablon tanımları bir başlığı izler.*
    - *template <typename T> gibi*
  - *T parametresi, üye fonksiyonlarda ve üye verilerde bir veri tipi olarak kullanılır.*
  - *İlave parametreler, virgül ile ayrılan listede belirtilir.*
    - *template <typename T1, typename T2> gibi*

Sınıf Şablonları (Class Templates)

- Stack sınıfının beyanı verilmektedir. Yığın yapısını sağlayan bu sınıf, farklı tipteki veriler için kullanılabilmektedir.

```
#include <iostream>
using namespace std;
template< typename T >
class Stack
{
public:
   Stack( int = 10 ); // default constructor (Stack size 10)

   // destructor
   ~Stack()
   {
      delete [] stackPtr; // deallocate internal space for Stack
   } // end ~Stack destructor

   bool push( const T & ); // push an element onto the Stack
   bool pop( T & ); // pop an element off the Stack

   // determine whether Stack is empty
   bool isEmpty() const
   {
      return top == -1;
   } // end function isEmpty

   // determine whether Stack is full
   bool isFull() const
   {
      return top == size - 1;
   } // end function isFull

private:
   int size; // # of elements in the stack
   int top; // location of the top element (-1 means empty)
   T *stackPtr; // pointer to internal representation of the Stack
}; // end class template Stack
```

Sınıf Şablonları (Class Templates)

```
// constructor template
template< typename T >
Stack< T >::Stack( int s )
   : size( s > 0 ? s : 10 ), // validate size
     top( -1 ), // Stack initially empty
     stackPtr( new T[ size ] ) // allocate memory for elements
{
   // empty body
} // end Stack constructor template

// push element onto Stack;
// if successful, return true; otherwise, return false
template< typename T >
bool Stack< T >::push( const T &pushValue )
{
   if ( !isFull() )
   {
      stackPtr[ ++top ] = pushValue; // place item on Stack
      return true; // push successful
   } // end if

   return false; // push unsuccessful
} // end function template push
```

Sınıf Şablonları (Class Templates)

```
// pop element off Stack;
// if successful, return true; otherwise, return false
template< typename T >
bool Stack< T >::pop( T &popValue )
{
   if ( !isEmpty() )
   {
      popValue = stackPtr[ top-- ]; // remove item from Stack
      return true; // pop successful
   } // end if

   return false; // pop unsuccessful
} // end function template pop
Stack

int main()
{
   Stack< double > doubleStack( 5 ); // size 5
   double doubleValue = 1.1;

   cout << "Pushing elements onto doubleStack\n";

   // push 5 doubles onto doubleStack
   while ( doubleStack.push( doubleValue ) )
   {
      cout << doubleValue << ' ';
      doubleValue += 1.1;
   } // end while
```

Sınıf Şablonları (Class Templates)

```
   cout << "\nStack is full. Cannot push " << doubleValue
      << "\n\nPopping elements from doubleStack\n";

   // pop elements from doubleStack
   while ( doubleStack.pop( doubleValue ) )
      cout << doubleValue << ' ';

   cout << "\nStack is empty. Cannot pop\n";

   Stack< int > intStack; // default size 10
   int intValue = 1;
   cout << "\nPushing elements onto intStack\n";

   // push 10 integers onto intStack
   while ( intStack.push( intValue ) )
   {
      cout << intValue++ << ' ';
   } // end while

   cout << "\nStack is full. Cannot push " << intValue
      << "\n\nPopping elements from intStack\n";

   // pop elements from intStack
   while ( intStack.pop( intValue ) )
      cout << intValue << ' ';

   cout << "\nStack is empty. Cannot pop" << endl;
} // end main
```

ÇIKTI:

```
Pushing elements onto doubleStack
1.1 2.2 3.3 4.4 5.5
Stack is full. Cannot push 6.6

Popping elements from doubleStack
5.5 4.4 3.3 2.2 1.1
Stack is empty. Cannot pop

Pushing elements onto intStack
1 2 3 4 5 6 7 8 9 10
Stack is full. Cannot push 11

Popping elements from intStack
10 9 8 7 6 5 4 3 2 1
Stack is empty. Cannot pop
```

Kaynaklar

- T.C. Lethbridge and R. Laganiere, Object-Oriented Software Engineering - Practical software development using UML and Java, McGraw Hill, Second Edition, 2005.
- H.M.Deitel and P.J.Deitel, C++ How To Program, 9E, Pearson Press, 2014.
- B. Stroustrup, The C++ Programming Language, 3rd Edition, Special Edition, Addison Wesley, 2000.
- Dr. Feza Buzluca, Ders Notları.
- Ç. Turhan ve F.C. Serçe, C++ Dersi: Nesne Tabanlı Programlama, 2nci Baskı, 2014.