PARAMETRİK ÇOK ŞEKİLLİLİK:

TEMPLATES

Binnur Kurt
kurt@cs.itu.edu.tr

Bilgisayar Mühendisliği Bölümü
İstanbul Teknik Üniversitesi

Sınıf Yapısı

Kalıtım

Çok Şekillilik

Templates

Parametrik Çok Şekillilik Nedir ?

Sınıflardaki fonksiyonların gövdeleri incelendiğinde, çoğu zaman yapılan işlemler, üzerinde işlem yapılan verinin tipinden bağımsızdır. Bu durumda fonksiyonun gövdesi, verinin tipi cinsinden, parametrik olarak ifade edilebilir:

```
int abs(int n) {
    return (n<0) ? -n : n;
}
long abs(long n) {
    return (n<0) ? -n : n;
}
float abs(float n) {
    return (n<0) ? -n : n;
}
```


- **C**

  abs(), fabs(), fabsl(), labs(), cabs(), ...

- **C++**

  Fonksiyonlara işlev yükleme ?

  Hata !

  Fonksiyon gövdeleri değişmiyor !

```
#include <iostream.h>
template <class T>
T abs(T n)
{
  return (n < 0) ? -n : n;
}
void main()
{
  int int1 = 5;
  int int2 = -6;
  long lon1 = 70000L;
  long lon2 = -80000L;
  double dub1 = 9.95;
  double dub2 = -10.15;
  // calls instantiate functions
  cout << "abs(" << int1 << ")=" << abs(int1) << endl;       // abs(int)
  cout << "abs(" << int2 << ")=" << abs(int2) << endl;       // abs(int)
  cout << "abs(" << lon1 << ")=" << abs(lon1) << endl;       // abs(long)
  cout << "abs(" << lon2 << ")=" << abs(lon2) << endl;       // abs(long)
  cout << "abs(" << dub1 << ")=" << abs(dub1) << endl;       // abs(double)
  cout << "abs(" << dub2 << ")=" << abs(dub2) << endl;       // abs(double)
}
```



```
template <class T>
void printArray(T *array,const int size){
      for(int i=0;i < size;i++)
          cout << array[i] << " " ;
cout << endl ;
}
```

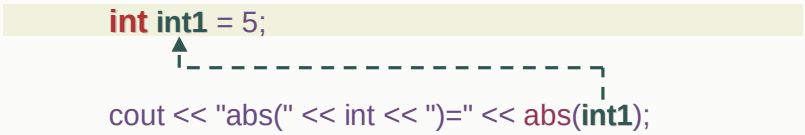
```
main() {
        int         a[3]={1,2,3} ;
        double      b[5]={1.1,2.2,3.3,4.4,5.5} ;
        char        c[7]={‘a’, ‘b’, ‘c’, ‘d’, ‘e’ , ‘f’, ‘g’} ;

        printArray(a,3)         ;
        printArray(b,5)         ;
        printArray(c,7)         ;
   return 0 ;
}
```



```
void printArray(int *array,cont int size){
        for(int i=0;i < size;i++)
            cout << array[i] << " " ;
    cout << endl ;
   }
void printArray(char *array,cont int size){
        for(int i=0;i < size;i++)
            cout << array[i] << "" ;
    cout << endl ;
   }
```

template'in İşleyişi

Gerçekte derleyici template ile verilmiş fonksiyon gövdesi için herhangi bir kod üretmez. Çünkü template ile bazı verilerin tipi parametrik olarak ifade edilmiştir. Verinin tipi ancak bu fonksiyona ilişkin bir çağrı olduğunda ortaya çıkacaktır. Derleyici her farklı tip için yeni bir fonksiyon oluşturacaktır. template yeni fonksiyonun verinin tipine bağlı olarak nasıl oluşturulacağını tanımlamaktadır.



programı ister template yapısı ile oluşturalım ister de template yapısı olmaksızın oluşturalım, programın bellekte kaplayacağı alan değişmeyecektir. Değişen, kaynak kodun boyu olacaktır. template yapısı kullanılarak oluşturulan programın kaynak kodu, daha anlaşılır ve hata denetimi daha yüksek olacaktır. Çünkü template yapısı kullanıldığında değişiklik sadece tek bir fonksiyon gövdesinde yapılacaktır.

template Parametresi bir Nesne Olabilir

```
class ComplexT{                    /* A class to define complex numbers */
   float re,im;
  public:
          :   // other member functions
   bool operator>(const ComplexT&) const ;    // header of operator> function
};

/* The Body of the function for operator + */
bool ComplexT::operator>(const ComplexT& z) const
{
  float f1 = re * re + im * im;
  float f2 = z.re * z.re + z.im * z.im;
  if (f1 > f2) return true;
     else      return false;
}
```



```
// template function
template <class type>
const type & MAX(const type &v1, const type & v2)
{
   if (v1 > v2) return v1;
      else      return v2;
}

void main()
{
  int i1=5, i2= -3;
  char c1='D', c2='N';
  float f1=3.05, f2=12.47;
  ComplexT z1(1.4,0.6), z2(4.6,-3.8);
  cout << MAX(i1,i2) << endl;
  cout << MAX(c1,c2) << endl;
  cout << MAX(f1,f2) << endl;
  cout << MAX(z1,z2) << endl;
}
```

Çoklu template Parametreli Argümanlar

```
// function returns index number of item, or -1 if not found template
template <class atype>
int find(const atype *array, atype value, int size) {
   for(int j=0; j<size; j++)
      if(array[j]==value) return j;
   return -1;
}
char chrArr[] = {'a', 'c', 'f', 's', 'u', 'z'};    // array
char ch = 'f'; // value to find
int intArr[] = {1, 3, 5, 9, 11, 13};
int in = 6;
double dubArr[] = {1.0, 3.0, 5.0, 9.0, 11.0, 13.0};
double db = 4.0;
```



```
void main()
{
   cout << "\n 'f' in chrArray: index=" << find(chrArr, ch, 6);
   cout << "\n 6 in intArray: index=" << find(intArr, in, 6);
   cout << "\n 4 in dubArray: index=" << find(dubArr, db, 6);
}
```

template İmzaları

```
template <class T>
void swap(T& x, T& y) {
      T temp ;
      temp = x ;
      x = y ;
      y = temp ;
}
char str1[100], str2[100] ;
int i,j ;
complex c1,c2;
swap( i , j ) ;
swap( c1 , c2 ) ;
swap( str1[50] , str2[50] ) ;
swap( i , str[25] ) ;
swap( str1 , str2 ) ;
```


Çoklu template Parametreli Yapılar

Template parametre sayısı birden fazla olabilir:

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

Bu durumda, derleyici sadece farklı dizi tipleri için değil aynı zamanda aranan elemanın farklı tipte olması durumunda farklı bir kod üretecektir:

```
short int result,si=100;
int invalue=5;
result = find(intArr, invalue,si) ;
```

Sınıf Template Yapısı

```
class Stack {
    int st[MAX];          // array of ints
    int top;              // index number of top of stack
  public:
    Stack();              // constructor
    void push(int var);   // takes int as argument
    int pop();            // returns int value
};

class LongStack {
    long st[MAX];         // array of longs
    int top;              // index number of top of stack
  public:
    LongStack();          // constructor
    void push(long var); // takes long as argument
    long pop();           // returns long value
};
```



```
template <class Type>
class Stack{
    enum {MAX=100};
    Type st[MAX];          // stack: array of any type
    int top;               // number of top of stack
  public:
    Stack(){top = 0;}      // constructor
    void push(Type );      // put number on stack
    Type pop();            // take number off stack
};
template<class Type>
void Stack<Type>::push(Type var) // put number on stack
{
  if(top > MAX-1)          // if stack full,
    throw "Stack is full!";// throw exception
  st[top++] = var;
}
```

```
template<class Type>
Type Stack<Type>::pop()          // take number off
stack
{
  if(top <= 0)                   // if stack empty,
      throw "Stack is empty!";   // throw exception
  return st[--top];
}
void main()
{
  // s1 is object of class Stack<float>
  Stack<float> s1;
  // push 2 floats, pop 2 floats
  try{
    s1.push(1111.1);
    s1.push(2222.2);
    cout << "1: " << s1.pop() << endl;
    cout << "2: " << s1.pop() << endl;
  }
  // exception handler
  catch(const char * msg) {
     cout << msg << endl;
  }
  // s2 is object of class Stack<long>
  Stack<long> s2;
  // push 2 longs, pop 2 longs
  try{
    s2.push(123123123L);
    s2.push(234234234L);
    cout << "1: " << s2.pop() << endl;
    cout << "2: " << s2.pop() << endl;
  }
  // exception handler
  catch(const char * msg) {
     cout << msg << endl;
  }
}     // End of program
```