BM102
Algoritma ve Programlama II

Şablonlar

İçerik

1. Fonksiyon Şablonu
2. Sınıf Şablonu
3. Şablon Parametreleri
4. Sınıf Şablon Özelleştirme

Çözümlü Sorular

Hedefler

- Şablon kavramının kullanım alanını açıklama
- Verilen fonksiyonun farklı veri tipindeki parametrelerle çalışması için fonksiyon şablonunu yazma
- Verilen fonksiyon şablonlarına yükleme yapma
- Sınıf şablonu tanımlama
- Sınıf şablon özelleştirme tanımı yapma
- Verilen fonksiyon tanımlamalarında hangi fonksiyon şablonunun kullanılacağını gösterme

Şablonlar

- *İng. Templates*
- *Şablonlar* aynı kod parçasını, farklı veri tipleri ile kullanılabilmemizi sağlayan bir yöntemdir.
  - fonksiyon şablonları
  - sınıf şablonları

1. Fonksiyon Şablonu

- *İng. Function template*
- Aynı fonksiyonun farklı tipteki parametrelerle çalışmasını *fonksiyon şablonu* kullanarak sağlayabiliriz.

```
template <class ŞablonParametresi>
ŞablonParametresi FonksiyonİsmiParametre Listesi)
{ ... }
```

- Şablonlarda parametrelerin tiplerini belirtmeyip yerine şablon parametresini veri tipi olarak kullanırız

1. Fonksiyon Şablonu…

Örnek 15.1

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

template <class T>
T topla(T deger1, T deger2)      // Fonksiyon şablonu
{
    return deger1+deger2;
}
int main()
{   int x=2,y=3;
    float a=2.1,b=3.5;
    cout<<"int tipiyle cagirma: "<< topla(x,y)<<endl;
    cout<<"float tipiyle cagirma: "<< topla(a,b)<<endl;
    return 0;
}
```

Şablon parametresi

Çıktı

```
int tipiyle cagirma: 5
float tipiyle cagirma: 5.6
```

1. Fonksiyon Şablonu…

- **Fonksiyon Şablonlarına Yükleme** de yapabiliriz.

Örnek 15.2

```
#include <iostream>
using namespace std;
template <class A>
A enBuyuk(A bir, A iki)              // Fonksiyon şablonu yükleme-2 parametreli
{
    if (bir>iki)
        return bir;
    else
        return iki;
}
template <class A>
A enBuyuk(A bir, A iki, A uc)        // Fonksiyon şablonu yükleme-3 parametreli
{
    return enBuyuk(enBuyuk(bir,iki),uc);
}
int main()
{
    cout<<"1.cagirim: "<<enBuyuk(5.4,3.1)<<endl;      // 1. fonksiyon çağırılır
    cout<<"2.cagirim: "<<enBuyuk(10,30,20)<<endl;     // 2. fonksiyon çağırılır
    cout<<"3.cagirim: "<<enBuyuk("veli","ali","ayse")<<endl;
                                                      // 2. fonksiyon çağırılır
    return 0;
}
```

Çıktı

```
1. cagirim: 5.4
2. cagirim: 30
3. cagirim: veli
```

2. Sınıf Şablonu

- *İng. Class templates*
- Sınıfa ait veri ve fonksiyon üyelerinin farklı veri tipleri ile kullanımını sağlar.
- Bir sınıf şablonu herhangi bir sınıf tanımı gibi yapılır.
- Ancak sınıf başlığından hemen önce, fonksiyon şablonlarında olduğu gibi şablon ve şablon parametreleri ile tanımlanır.

```
template <class ŞablonParametresi>
    class SınıfİsmI{...}
```

2. Sınıf Şablonu…

Örnek 15.3

```
#include <iostream>
using namespace std;
template <class T>
class Islem                                          // Sınıf şablonu
{
    T x, y;
    public:
       Islem(T _x, T _y):x(_x),y(_y){}  // Yapıcı fonksiyon
       T kucukSayi();          // Üye fonksiyon prototipi
};
template <class T>
T Islem<T>::kucukSayi()     // Parametrelerin küçüğünü döndüren üye fonksiyon
{
    if (x < y){
        return x;
    }
    return y;
}
```

```
int main ()
{
    Islem <int> islem1(100, 75);
    cout << islem1.kucukSayi()<<endl;
    Islem <double> islem2(3.24, 8.1);
    cout << islem2.kucukSayi()<<endl;
    return 0;
}
```

Çıktı

```
75
3.24
```

3. Şablon Parametreleri

- Şablon parametreleri bir veya daha fazla olabilir. Birden fazla şablon parametresi “,” (virgül) işareti ile aşağıda gösterildiği gibi tanımlanır.

```
template <class T1,class T2, ... >
```

3. Şablon Parametreleri...

Örnek 15.4

```
#include <iostream>
using namespace std;
template <class T1, class T2>
class Islem          // İki parametreli sınıf şablonu
{
    T1 x;
    T2 y;
    public:
        Islem(T1 _x, T2 _y):x(_x),y(_y){}      // Yapıcı Fonksiyon
        T2 topla();                             // Üye fonksiyon prototipi
};
template <class T1, class T2>
T2 Islem<T1,T2>::topla()          // x ve y üyelerinin toplamını döndüren üye fonksiyon
{
    return  x + y;
}
int main ()
{
    Islem <int, double> islem(100, 3.4);
    cout << islem.topla();
    return 0;
}
```

Çıktı

```
103.4
```

3. Şablon Parametreleri...

Örnek 15.5

```
#include <iostream>
using namespace std;
template <class T, int N>          // Veri tipi ve değer şablon parametreleri
class Islem                        // Sınıf şablonu
{
    T dizi[N];                     // N elemanlı T tipinde dizi tanımı
    public:
        void elemanEkle(int i, T eleman);
         T elemanAl(int i);
};
template <class T, int N>
void Islem<T,N>::elemanEkle(int i, T eleman) // dizinin i. elemanına atama yapılır
{
    dizi[i] = eleman;
}
template <class T, int N>
T Islem<T,N>::elemanAl(int i)
{
    return dizi[i];
}
```

```
int main ()
{
    Islem <int,5> islem1;
    Islem <double,10> islem2;
    islem1.elemanEkle (0,100);
    islem2.elemanEkle (3,6.9);
    cout<<islem1.elemanAl(0)<<endl;
    cout<<islem2.elemanAl(3)<<endl;
    return 0;
}
```

Çıktı

```
100
6.9
```

4. Sınıf Şablon Özelleştirme

- *İng. Class template specialization*
- Şablonlar, aynı kod parçasının farklı veri tipleri için kullanılabilmesine olanak sağlar.
- Ancak şablonların kullanımı sırasında kimi zaman farklı veri tipleri için kodun bir bölümünde veri tipine göre değişiklik yapılması gerekebilir.
- Bu durumlarda *sınıf şablon özelleştirme* yönteminden yararlanılır.

4. Sınıf Şablon Özelleştirme…

Örnek 15.6

```
#include <iostream>
using namespace std;
template <class T>
class Islem                                  // Sınıf şablonu
{
    T sayi;
    public:
        Islem(T _sayi): sayi(_sayi){}        // Yapıcı fonksiyon
        T artir()                            // Sayıyı bir arttıran fonksiyon
        {
            return ++sayi;
        }
};
template <>
class Islem<char>                            // Sınıf şablon özelleştirmesi
{
    char ch;
    public:
        Islem (char _ch): ch(_ch){}          // Yapıcı fonksiyon
        char buyukHarf()                     // ch karakterini büyük harfe çeviren fonksiyon
        {
           if ((ch>='a')&&(ch<='z'))
                ch += -32;                   //'A'-'a'=-32
           return ch;
        }
};
```

```
int main () {
    Islem<int> islem1(5);
    Islem<char> islem2('e');
    cout << islem1.artir()<< endl;
    cout << islem2.buyukHarf()<< endl;
    return 0;
}
```

Çıktı

```
6
E
```

Çözümlü Sorular

Soru

6. Matematikte sıralı çift (ordered pair) (x,y) formatında yazılmış iki değerden oluşur. Aşağıda sadece tamsayı tipinde x ve y üyelerine sahip Cift isimli sınıf verilmiştir. Bu sınıfı x ve y herhangi bir veri tipinde olabilecek şekilde sınıf şablonu olarak tekrar yazınız. Cift tipinde içeriği (2.5,'c') ve ("Ali",3000) olan iki nesne yaratıp nesneleri ekrana yazdıran bir main() fonksiyonu yazınız.

```
class Cift
{
    int bir,iki;
    public:
        Cift(int,int);
        void yaz();
};
Cift::Cift(int a,int b):bir(a),iki(b){}
void Cift::yaz()
{
    cout<<"("<<bir<<","<<iki<<")"<<endl;
}
```

Çözümlü Sorular

Cevap

6.

```
#include <iostream>
using namespace std;
template <class A,class B>
class Cift
{   A bir;
    B iki;
    public:
        Cift(A,B);
        void yaz();
};
template <class A,class B>
Cift<A,B>::Cift<A,B>(A a,B b):bir(a),iki(b){}

template <class A,class B>
void Cift<A,B>::yaz()
{   cout<<"("<<bir<<","<<iki<<")"<<endl;
}
int main()
{   Cift<float,char> x(2.5,'c');
    Cift<string,int> y("Ali",3000);
    x.yaz();
    y.yaz();
    return 0;
}
```

Çözümlü Sorular

Soru…

8. Katsayıları herhangi bir veri tipi olan polinomları yaratabileceğimiz bir `Polinom` sınıf şablonu geliştirelim. Polinomlar aşağıda görüldüğü gibi terimlerden oluşur ve her terimde bir katsayı ve üs yer almaktadır. Örneğin, $2x^4$ teriminin katsayısı 2, üssü ise 4'tür.

$$a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0,$$

$a_0,..a_n$: katsayılar
n: polinomun derecesi (polinomdaki en büyük üs)

Bir polinomu terimler dizisi olarak saklayabiliriz. Yaratacağınız `Polinom` sınıf şablonu aşağıdaki bileşenlerden oluşacaktır:

- Veri üyeleri: derece, katsayı dizisi (üsleri indeks, katsayıları da elemanlar olarak alabilirsiniz.
  Örneğin,
  $2x^3 + 4x + 5$ polinomunu aşağıdaki dizide saklayabiliriz.

0	1	2	3
5	4	0	2

- Üye fonksiyonlar
  - Yapıcı fonksiyon: Polinomun derecesini parametre olarak alır ve dinamik bir dizi yaratır.
  - >> operatör yükleme fonksiyonu: dizinin içine katsayıları okur.
  - = operatör yükleme fonksiyonu: bir polinom nesnesini başka bir nesneye atar.
  - << operatör yükleme fonksiyonu: polinomu yazdırır.

Çözümlü Sorular

Soru

main() aşağıda verilmiştir.

```
int main()
{
    Polinom<int> a,b;
    cin>>a;
    cin>>b;
    cout<<"Polinomlar:"<<endl;
    cout<<a;
    cout<<b;
    b=a;
    cout<<"Atamadan sonra:"<<endl;
    cout<<b;
    return 0;
}
```

Örnek Çıktı

```
Polinomun derecesi: 3
x3 katsayı: 2
x2 katsayı: 0
x1 katsayı: 4
x0 katsayı: 5
Polinomun derecesi: 2
x2 katsayı: 3
x1 katsayı: 0
x0 katsayı: 7
Polinomlar:
2x3+4x1+5x0
3x2+7x0
Atamadan sonra:
2x3+4x1+5x0
```

Çözümlü Sorular

Cevap…

```
// ---------- Polinom.h ----------
#include <iostream>
using namespace std;
template <class T>
class Polinom
{   int derece;
    T* dizi;
    public:
        Polinom(){}
        Polinom(int _derece, T* _dizi):derece(_derece),dizi(_dizi){}
        void operator=(const Polinom<T>& x)
        {
            if (derece != x.derece){
                derece=x.derece;
                delete[] dizi;
                dizi=new T[derece+1];
            }
            for (int i=0;i<=derece;i++)
                dizi[i]=x.dizi[i];

        }
        friend std::ostream& operator<<(std::ostream&,
                                        const  Polinom<T>&);
        friend std::istream& operator>>(std::istream&, Polinom<T>&);
};
```

Çözümlü Sorular

Cevap…

```
// ---------------------- Uygulama.h ----------------------
#include <iostream>
#include "Polinom.h"
using namespace std;
template <class T>
std::ostream& operator<<(std::ostream& cout, const Polinom<T>& p)
{   for (int i=p.derece;i>=0;i--){
        if (p.dizi[i]!=0){
            cout<<p.dizi[i]<<"x"<<i;
            if (i>0) cout<<"+";
        }
    }
    cout<<endl;
}
```

Çözümlü Sorular

Cevap…

```
template <class T>
std::istream& operator>>(std::istream& cin, Polinom<T>& p)
{    cout<<"Polinomun derecesini girin:";
     cin>>p.derece;
     int* dizi=new int[p.derece+1];
     for (int i=p.derece;i>=0;i--){
          cout<<"x"<<i<<" katsayi:";
          cin>>dizi[i];
     }
     p.dizi = dizi;
     return cin;
}
```

Çözümlü Sorular

Cevap

```
int main()
{    Polinom<int> a,b;
     cin>>a;
     cin>>b;
     cout<<"Polinomlar:"<<endl;
     cout<<a;
     cout<<b;
     b=a;
     cout<<"Atamadan sonra:"<<endl;
     cout<<b;
     return 0;
}
```