5. Fonksiyonlar - II

Fonksiyonlara Parametrelerin Referans Olarak Gönderilmesi

Bir önceki derste fonksiyonlara pointerler kullanarak parametre aktarımı gerçekleştirilmişti. Bilindiği gibi pointerler bir değişkenin adresini tutan başka bir değişkendir. Aşağıda verilen programda

```
void degistir2( int *x, int *y );
```

şeklinde verilen fonksiyona dışardan iki değişkenin adresi gönderilmektedir. Bu adreslerin içinde yer alan değerler adreslerden yararlanarak yer değişmektedir.

C++'da ampersand (&) simgesini kullanarak değişkenler için referanslar tanımlamak mümkündür. Bir değişken için referans tanımlanması aynı bellek gözüne ikinci bir isim vermek demektir. Referans, bir başka ifade ile bir değişkenin diğer ismi gibi davranan bir göstericidir.

degistir2 fonksiyonu

```
void degistir3( int& x, int& y ) ;
```

şeklinde tanımlanırsa x ve y kendisine gönderilen değişken ile aynı bellek alanını kullanacaktır. Böylece x ve y de yapılan değişiklik kendisine gönderilen parametre değişkenlerin değerini(içeriğini) değiştirecektir. Referans ile parametre aktarımı yapılan bir fonksiyonun çağrılması yazım şekli olarak değer ile çağırmada olduğu gibi yapılabilmektedir. Böylece yazım ve kullanımda kolaylık sağlamaktadır.

C'de anaprogramdan parametre olarak gönderilen bir verinin orijinal değerinin fonksiyon içinde değişmesi isteniyorsa o verinin adresinin parametre olarak fonksiyona gönderilmesi gerekir.

Büyük boyutlu veriler bir fonksiyona parametre olarak aktarılırken değerleri yerine adreslerinin aktarılması tercih edilir. Çünkü değer olarak aktarılması durumunda büyük boyutlu verinin tamamı yığına kopyalanacaktır. Bu da hem bellekte fazla yer harcanmasına hem de programın yavaşlamasına neden olacaktır.

Yerel değişkenler referans ile döndürülemez!. Yerel değişkenler sadece tanımlandıkları bloğun içinde veya bir fonksiyonun içinde tanımlandıklarından fonksiyondan çıktıktan sonra bellekten atılırlar.

```
int& fonk(int a, int b ) // Referans ile veri geri döndürülecek
{ int x; // Yerel değişken.
  x=a+b;
  return x; // Hata! x'nin adresi artık yok.
}

İnt* fonk(int a, int b ) // Adres ile veri geri döndürülecek
{ int x; // Yerel değişken.
  x=a+b;
  return &x; // Hata! x'nin adresi artık yok.
}
```

Bu fonksiyonun doğru olanı yerel değişkenlerin değer ile geri dönderilmesidir..

```
int fonk(int a, int b ) // Değer geri döndürülecek
{ int x; // Yerel değişken.
  x=a+b;
  return x; // Doğru! x'nin değeri dönderiliyor
}
```

Aşağıda verilen programı inceleyiniz.

```
#include <iostream.h>
#include <conio.h>
//Pointer ile parametre aktarımı
void degistir2( int *x, int *y )
{       int gecici;
        gecici = *x;
        *x = *y;
        *y = gecici;
}
//Referans ile parametre aktarımı
void degistir3( int &x, int &y )
{       int gecici;
        gecici = x;
        x = y;
        y = gecici;
}
int main( )
{   int a, b;
    a = 12;
    b = 27;
    cout<< "degistir2 fonk dan once a : "<<a<<"  b : "<<b<<endl;
    degistir2(&a, &b);
    cout<< "degistir2 fonk dan sonra a : "<<a<<"  b : "<<b<<endl;

    cout<< "degistir3 fonk dan once a : "<<a<<"  b : "<<b<<endl;
    degistir3(a, b);// Argumanları aktarırken, baslarina & konur
    cout<< "degistir3 fonk dan sonra a : "<<a<<"  b : "<<b<<endl;
    getch();
    return 0;
}
```

```
degistir2 fonk dan once a : 12  b : 27
degistir2 fonk dan sonra a : 27  b : 12
degistir3 fonk dan once a : 27  b : 12
degistir3 fonk dan sonra a : 12  b : 27
```

int fonk(const int &a, const int &b) şeklinde tanımlanan fonksiyonda a ve b sabit referanslardır. Dolayısı ile fonksiyon içinde değerleri değiştirilemez.

```
int fonk(const int &a, const int &b )
{ int x;  x=a+b;
  a++; b++;     //hata çünkü sabit referansın değeri değiştirilemez
  return x;
}
```

Fonksiyon Parametrelerin varsayılan başlangıç değerleri

Fonksiyonlar tanımlanırken parametrelerin varsayılan başlangıç değerleri de verilebilir. Böylece fonksiyon çağrıldığında parametre kullanılmamış ise kullanılmayan parametrenin varsayılan değeri fonksiyon için de kullanılır. Parametre değeri verilmiş ise bu takdirde bu değer fonksiyon içinde kullanılacaktır.

Aşağıda ki örnek programı inceleyiniz.

```
#include <iostream.h>
#include <conio.h>

int divide (int a=8, int b=2)
{
   int r;
   r=a/b;
   return (r);
}
int main ()
{
   cout << divide ( )<<endl;
   cout << divide (12)<<endl;
   cout << divide (21,3)<< endl;
   getch();
   return 0;
}
```

```
4
6
7
```

divide ( ) şeklinde fonksiyon çağrıldığında a değişkeni 8 ve b değişkeni 2 değerini alacak ve fonksiyonda buna göre değerler hesaplanacaktır. Sonuç 4 olacaktır.

divide (12) şeklinde fonksiyon çağrıldığında a değişkeni 12 ve b değişkeni 2 değerini alacak ve fonksiyonda buna göre değerler hesaplanacaktır. Sonuç 6 olacaktır.

divide (20,4) şeklinde fonksiyon çağrıldığında ise a=21 ve b=3 değerini alacak ve fonksiyonda buna göre değerler hesaplanacaktır. Sonuç 7 olacaktır.

Aşağıda verilen fonksiyon üç değişik parametre almaktadır. b ve c parametreleri başlangıç değeri olarak 2 ve 3 değerlerini almaktadır.

```
void Topla (int a, int b=2, int c=5) { … }
```

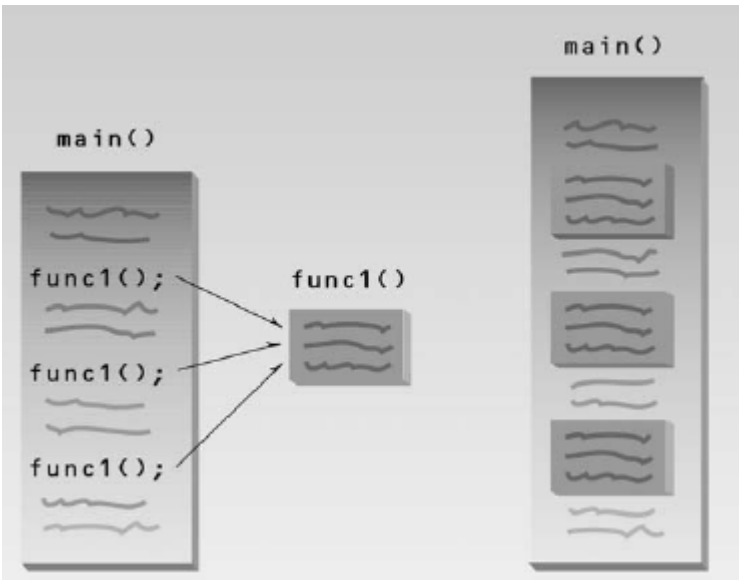
Bu fonksiyon aşağıdaki gibi üç farklı şekilde çağrılabilir:

Topla(5); //a=5 b=2 c=5
Topla(5,6); //a=5 b=6 c=5
Topla(5,6,7) //a=5 b=6 c=7
Topla(7, , 4); // Hata ikinci parametre eksik bırakılmış. Parametre atlanmamalı sırası ile verilmelidir.

inline fonksiyonlar(Makrolar)

C dilinde makroları tanımlamak için önişlemcinin #define direktifi kullanılmaktadır. C++ da ise makrolar normal fonksiyonlar gibi yazılabilmektedir. Ana program içinden bir fonksiyon ismi ile karşılaşıldığında (fonksiyon çağrıldığında) fonksiyonun bellekte yerleştiği alana bir dallanma gerçekleştirilir. Bu sırada dönüş adresi ve bazı işlemci kaydedici içerikleri yığına saklanır. Fonksiyon tanımında yer alan komutlar icra edildikten sonra yığından dönüş adresi ve kaydedici içerikleri alınarak ana programda kalınan noktadan devam edilir. Fonksiyon her çağrıldığında bu işlem tekrar edilir. Görüldüğü üzere fonksiyon bellekte bir kez yer kaplamaktadır. Ancak her fonksiyon çağrısında yığına değer atma ve çekme gibi fazladan işlemler yapılmaktadır. Bu ise zaman kayıplarına neden olmaktadır. Zaman kısıdının önemli olduğu uygulamalarda bu önem arzeder.

Makrolarda ise, çağrının yapıldığı her yere makronun kodu eklenir. Bu durumda her makro çağrısı programın boyunun uzamasına neden olacaktır. Ancak fonksiyonu çağırma yani başka bir koda gitme, geri gelme ve parametre aktarımı olmadığından makro çağrıları normal fonksiyon çağrılarına göre daha daha hızlı gerçekleşecektir. Özellikle hızlı çalışmasını istediğimiz ve kodu kısa olan program parçalarını makro olarak oluşturmak doğru olacaktır. Şekil 4.1 de normal ve makro olarak çağrılan fonksiyonun bellekteki durumları görülmektedir.



Şekil 4.1 a) Normal fonksiyon çağrısı b) makro çağrısı

Bir fonksiyonu marka olarak tanımlamak için Fonksiyon tanımının başına **inline** ifadesi yazılır.

```
inline int kare (int a)
{
   return a*a;
}
```