Fonksiyonların Varsayılan Argüman Alması

Öyle fonksiyonlar vardır ki bunlara yapılan çağrılarda fonksiyonun bazı parametrelerine çok büyük çoğunlukla hep aynı değerler gönderilir. Fonksiyona alışılmış değerlerin değil de farklı değerlerin gönderilmesi de söz konusu olabilir; ancak bu durum çok daha seyrek oluşmaktadır. Böyle fonksiyonlara yapılan çağrılarda programcı, parametrelere her zaman alışılmış aynı değerleri gönderecek olmasına karşın yine de her fonksiyon çağrısında aynı argüman ifadeleri yazmak zorundadır. Bu durum öncelikle programcı üzerinde gereksiz bir yük oluşturur. Ayrıca her zaman gönderilen argümanların fiilen fonksiyon çağrısında yazılması sırasında, programcı fonksiyona yanlışlıkla farklı değer(ler) de gönderilebilir. Örneğin programcı iki argümanın sırasını karıştırabilir. Alışılmış değerlerin fonksiyona derleyicinin üretiği kodla gönderilmesi programcının işini kolaylaştırır, hata yapma riskini azaltır. Varsayılan argüman kullanımı kodun okunmasını da kolaylaştırır.

C dilinde bir fonksiyonun kaç tane parametre değişkeni varsa fonksiyone o kadar sayıda argüman *(argument - actual parameter)* gönderilmelidir. Oysa C++ dilinde bir fonksiyon, parametre değişkeni sayısından daha az sayıda argümanla çağrılabilir. Bir fonksiyonun bir ya da birden fazla parametre değişkeni varsayılan argüman *(default argument)* alabilir. Bunun anlamı şudur: Fonksiyon çağrısı ile fonksiyonun parametre değişkenine bir değer gönderilmemiş ise derleyici argüman olarak daha önce belirlenen bir ifadeyi kullanır.

Bir parametre değişkeninin önceden belirlenmiş bir değeri alabileceği fonksiyonun bildiriminde ya da fonksiyonun tanımında, parametre değişkeninden sonra eşittir *(=)* atomundan sonra yazılan bir ifadeyle belirtilir. Aşağıdaki kodu inceleyin:

```
#include <iostream>

void foo(int x = 10, int y = 20, int z = 30);

void foo(int x, int y, int z)
{
 std::cout << "x = " << x << "\n";
 std::cout << "y = " << y << "\n";
 std::cout << "z = " << z << "\n";
 std::cout << "-------------------\n";
}

int main()
{
 foo();   //foo(10, 20, 30)
 foo(100);  //foo(100, 20, 30)
 foo(100, 200);  //foo(100, 200, 30)
 foo(100, 200, 300);     //foo(100, 200, 300)
}
```

- Yukarıdaki kodda *foo* fonksiyonunun bildiriminde varsayılan argümanlar kullanılıyor. *main* fonksiyonu içinde yapılan ilk çağrıda foo fonksiyonuna hiç bir argüman gönderilmiyor. Bu durumda parametre değişkenleri olan *x*, *y* ve *z*'ye varsayılan değerler olan *10, 20* ve *30* değerleri aktarılır.
- Yapılan ikinci çağrıda, *foo* fonksiyonuna yalnızca *100* değeri gönderiliyor. Bu durumda birinci parametre değişkenine *100* değeri kopyalanırken, ikinci ve üçüncü parametrelerine sırasıyla *20* ve *30* değerleri kopyalanır.
- *foo* fonksiyonuna yapılan üçüncü çağrıda ise fonksiyona *100* ve *200* değerleri gönderiliyor. Bu durumda birinci parametreye 100, ikinci parametreye *200* değerleri kopyalanırken üçüncü parametreye *30* değeri kopyalanır.
- Dördüncü ve son çağrıda *foo* fonksiyonuna *100*, *200* ve *300* değerleri gönderiliyor. Bu durumda, parametre değişkenlerinden hiçbiri varsayılan bir değer almıyor.

Görüldüğü gibi varsayılan değerler fonksiyon çağrılırken fonksiyona gönderilmeyen argüman olan ifadelerin yerine geçmektedir

Daha soldaki parametrelere fonksiyon çağrısıyla argüman gönderilip, daha sağdaki diğer parametrelere argüman gönderilmeden fonksiyon çağrılırsa varsayılan değerler kullanılabilir. Ancak bunun tersi geçerli değildir. Aşağıdaki çağrı biçimi her durumda geçersizdir:

```
foo(, 10);
```

Bir parametre değişkeni için varsayılan bir değer belirlenmişse, bu parametre değişkeninin daha sağında bulunan parametre değişkenlerinin hepsi varsayılan değerler almak zorundadır:

```
void func(int x = 10, int y); //geçersiz!
void foo(int x, int y = 20); //Geçerli
```

Varsayılan değer belirlenmemiş olan bütün parametre değişkenlerine çağrı ifadesi ile gereken argümanlar gönderilmek zorundadır.

Varsayılan argüman kullanımı tamamen derleme zamanına *(compile time)* yönelik bir mekanizmadır. Varsayılan argümanlar derleme zamanında derleyici tarafından ele alınırlar. Varsayılan argümanların kullanılması programın çalışma zamanı açısından ek bir maliyet doğurmaz.

Bir fonksiyonun parametre değişkeni olan gösterici de varsayılan değer alabilir. Aşağıdaki örneği inceleyin:

```
#include <iostream>

void put_message(const char *p = "Success!")
{
 std::cout << p;
}

int main()
{
 put_message("Failed!");
 put_message();
}
```

Yukarıdaki programda put_message isimli fonksiyonun gösterici olan parametre değişkeni p için varsayılan argüman olarak "*Success!*" string sabiti *(string literal)* geçiliyor. *main* fonksiyonu içinde yapılan ilk çağrıda, fonksiyona "*Failed!*" string sabiti argüman olarak geçilirken, ikinci çağrıda fonksiyona herhangi bir argüman gönderilmiyor. Bu durumda varsayılan argüman olan "*Success*" string sabiti fonksiyona gönderilmiş olur. Yani yapılan ikinci çağrıyla ekrana "*Success!*" yazısı yazdırılır.

Varsayılan argüman olarak belirlenen ifade bir sabit ifadesi *(constant expression)* olmak zorunda değildir. Değişken içeren ifadeler de varsayılan argüman olarak kullanılabilir. Varsayılan argüman olarak kullanılan ifadelerde daha önce bildirimi yapılmış global değişkenler kullanılabileceği gibi fonksiyon çağrıları da yer alabilir.

```
int func1();
int func2(int);
int func3(double = 3.14);
int g = 10;
int func4(int a = func1(), int b = func2(g), int c = func3());
```

Yukarıda yapılan tüm fonksiyon bildirimleri geçerlidir. *func4* fonksiyonunun her üç parametresi de varsayılan argüman alıyor. Birinci parametre olan a değişkenine, fonksiyon çağrı ifadesi ile bir değer atanmaz ise, func1 fonksiyonunun geri dönüş değeri atanır. İkinci parametre değişkeni olan *b*'ye bir değer geçilmez ise, *func2* fonksiyonunun geri dönüş değeri atanır. Bu arada çağrılan func2 fonksiyonuna, global değişken olan *g* değişkeninin değeri geçilir. Son parametre değişkenine değer geçilmediği zaman ise, bu parametre değişkenine *func3* fonksiyonunun geri dönüş değeri atanır. Bu durumda çağrılacak *func3* fonksiyonu da kendisine argüman gönderilmediği için varsayılan değer olarak belirlenen *3.14* değerini alır. Varsayılan argümanlara ilişkin ifadelerin değerlendirilmesi fonksiyonun çağrıldığı noktada gerçekleşir. Yani *func4* fonksiyonu eğer çağrılmaz ise *func1*, *func2* ve *func3* fonksiyonları de çağrılmaz.

Varsayılan argüman olarak belirlenen değer yalnızca bir kez yazılmalıdır. Varsayılan argüman olan değerin hem fonksiyon bildiriminde, hem de fonksiyon tanımında yer alması geçersizdir:

```
void func(int x = 10, int y = 20);

void func(int x = 10, int y = 20) // geçersiz.
{
 //...
}
```

Hatırlayacağınız gibi, fonksiyon bildirimlerinde parametre değişkenlerinin isimleri yazılmak zorunda değildir. Varsayılan bir değer alacak parametre değişkeni için de isim yazılması zorunlu değildir. Aşağıdaki iki bildirim de geçerlidir:

```
void func(int a = 10, int b = 20);
void foo(int = 10, int = 20);
```

Ancak fonkiyonun varsayılan değer alacak parametre değişkeni bir gösterici ise ve fonksiyon bildiriminde parametre değişkeni olan göstericiye isim verilmiyorsa dikkatli olunmalıdır:

```
void func(char *= "Ahmet"); // Geçersiz!
```

Derleyici burada "="* karakterlerini tek bir atom olarak ele alıp işlemli atama operatörü olarak değerlendirir *(En uzun atom kuralı - maximum munch)*. Bildirim geçersizdir. Bu durumda varsayılan argümana ilişkin bu iki karakter bitişik yazılmamalıdır:

```
void func(char * = "Ahmet");
```

Gösterici parametre değişkenleri varsayılan argüman alabildiği gibi, referans parametre değişkenleri de varsayılan değerler alabilir:

```
int g = 20;

void func(const int &r = g);

int main()
{
 int y = 30;
 func();
 func(y);
 return 0;
}

// .cpp dosyası
#include <iostream>

void func(int &r)
{
 std::cout << r << "\n";
}
```

Bildiriminde varsayılan argüman belirtilmemiş bir fonksiyon için ilave bir bildirimle *(redeclaration)* varsayılan argüman belirtilebilir. Kullanacağımız bir kütüphanenin başlık dosyasından aşağıdaki gibi bir bildirim kaynak dosyamıza/dosyalarımıza gelmiş olsun:

```
void func(int, int, int);
```

Biz bu fonksiyona çağrı yapacağımız çağrılarda son parametreye hep 10 değerini gönderecek olalım. Aşağıdaki gibi bir "yeniden bildirim" yapabiliriz:

```
void func(int, int, int = 10);
```

Benzer şekilde yeni bir bildirimle varsayılan argüman sayısı arttırılabilir.

```
//xlib.h
void foo(int, int, int = 10);

//client.cpp
#include "xib.h"
void foo(int, int = 20, int);
```

foo fonksiyonu için yapılan ikinci bildirimde eğer fonksiyonun *3*. parametresi için de varsayılan argüman belirtilmiş

olsaydı sentaks hatası oluşurdu.

Şöyle bir fonksiyon bildirimi edinmiş olalım:

```
void f(int x, int y, int z);
```

Sıklıkla çağıracağımız bu fonksiyonu *2*. parametresine hep aynı değeri göndererek çağıracak olalım. Bunun için sarmalayıcı bir fonksiyon yazabiliriz:

```
void f_(int x, int z, int y = 10)
{
 f(x, y, z);
}
```

Varsayılar argümanların bazen tek varlık nedeni çağıran tarafın bir parametreye argüman gönderip göndermediğini test etmektir. Aşağıda tanımlanan *print_date* fonksiyonuna bakalım. Fonksiyon kendisine gün, ay ve yıl olarak gönderilen tarihi standart çıkış akımına yazdırıyor. Eğer belirli bir parametreye gün, ay ya da yıl değeri gönderilmez ise çağrının yapıldığı tarihin ilgili değerleri kullanılacak:

```
#include <ctime>
#include <cstdio>

void display_date(int d = -1, int m = -1, int y = -1);

void display_date(int d, int m, int y)
{
 if (y == -1) {
  std::time_t timer;
  std::time(&timer);
  auto p = std::localtime(&timer);
  y = p->tm_year + 1900;
  if (m == -1) {
   m = p->tm_mon + 1;
   if (d == -1) {
    d = p->tm_mday;
   }
  }
 }
 std::printf("%02d/%02d/%02d\n", d, m, y);
}

int main()
{
 display_date(11, 11, 2011);
 display_date(5, 8);
 display_date(4);
 display_date();
}
```