C'de Bağlanım (Linkage) Nedir, Ne Değildir?

Kaan Aslan
28 Nisan 2012

*C'*de kullanılan her değişken (*identifier*) için bir bildirimin yapılmış olması gerekir. Bağlanım (*linkage*) farklı bildirimlerdeki aynı isimli değişkenlerin aynı nesneyi ya da fonksiyonu belirtip belirtmediğini anlatan bir özelliktir. *C'*de değişkenlere ilişkin üç bağlanım durumu söz konusudur:

1. Dışsal bağlanım (*external linkage*)

2. İçsel bağlanım (*internal linkage*)

3. Bağlanıma sahip olmama (*no linkage*)

Eğer bir değişkenin belirttiği nesne ya da fonksiyona başka bir derleme birimindeki (*translation unit*) bir bildirimle erişilebiliyorsa bu değişkenler dışsal bağlanım (*external linkage*) özelliğine sahiptir.[1] Yani dışsal bağlanım farklı derleme birimlerindeki aynı isimle bildirilmiş değişkenlerin aynı nesneyi ya da fonksiyonu belirtmesi durumudur. Örneğin *"a.c"* dosyasındaki *x* değişkeni ile *"b.c"* dosyasındaki *x* değişkeni aynı nesneyi belirtiyorsa bu iki bildirimdeki *x* değişkeni dışsal bağlanıma (*external linkage*) sahiptir. Başka bir deyişle, dışsal bağlanıma sahip aynı isimli bütün değişken bildirimleri aslında aynı nesneyi ya da fonksiyonu belirtiyor durumdadır. Eğer bir *x* değişkeninin aynı derleme biriminin farklı yerlerinde yapılan bildirimleri aynı nesneyi ya da fonksiyonu belirtiyorsa fakat bu değişkenin belirttiği nesne ya da fonksiyona başka bir derleme birimindeki bir bildirimle erişilemiyorsa bu bildirimlerdeki değişkenler içsel bağlanıma (*internal linkage*) sahiptir. Yani içsel bağlanıma sahip bir değişkeni aynı derleme biriminin farklı faaliyet alanlarında kullanabiliriz, fakat farklı derleme birimlerinde kullanamayız. Nihayet, eğer bir değişkenin belirttiği nesne ya da fonksiyon başka bir faaliyet alanından yeni bir bildirimle kullanılamıyorsa bu değişkenin bağlanım özelliği yoktur (*no linkage*).

C'de (*C90*, *C99*, *C11*) *extern* belirleyicisi genel olarak, bildirilen değişkenin dışsal bağlanıma sahip olduğunu belirtmektedir. Örneğin:

```
extern int a;        /* a dışsal bağlanıma sahip */
extern foo(void);    /* foo dışsal bağlanıma sahip */
```

Fonksiyon bildirimlerindeki varsayılan bağlanım durumu dışsal bağlanımdır. Yani bir fonksiyon ister global düzeyde, isterse yerel düzeyde bildirilmiş olsun bildiriminde *extern* belirleyicisi kullanılmamış olsa bile bu belirleyicinin kullanılmış olduğu varsayılır. Örneğin, aşağıdaki bildirimin global ya da yerel düzeyde yapıldığını varsayalım:

```
void foo(void);
```

Bu bildirimlerin eşdeğeri şöyledir:

```
extern void foo(void);
```

Global nesne bildirimlerinin de varsayılan bağlanımı yine dışsal bağlanımdır. Fakat global nesne bildirimlerinde *extern* belirleyicisi kullanılmamışsa sanki bu belirleyici kullanılmış gibi bir etki oluşmaz. Örneğin aşağıdaki bildirimin global düzeyde yapıldığını varsayalım:

```
int g_a;
```

Burada *g_a* dışsal bağlanıma sahiptir, ancak bu bildirimin eşdeğeri aşağıdaki gibi değildir:

```
extern int g_a;     /* yukarıdaki bildirimin eşdeğeri değil */
```

Bildiğiniz gibi, *C*'de *extern* belirleyicisi ile bir nesne bildirimi yapıldığında bu bildirim bir tanımlama belirtmez. Yani bildirilen değişken için bellekte bir yer ayrılmaz.[2] Dolayısıyla *extern* belirleyicisi ile bildirilmiş olan bir nesnenin toplamda tek bir tanımlamasının bulunması gerekir. Bu tanımlama herhangi bir derleme biriminde yapılabilir.

*C'*de global bildirimlerdeki *static* belirleyicisi değişkenin içsel bağlanıma sahip olduğunu belirtir. *static* belirleyicisi ile bildirilen bir global nesne ya da fonksiyona başka bir derleme biriminden erişilemez. Yani *static* belirleyicisi değişkene içsel bağlanım özelliği vererek onun başka bir derleme biriminden kullanımını engellemektedir. Farklı iki derleme biriminde aynı isimli *static* global değişkenler bildirilebilir. Bu durumda bu değişkenler farklı nesne ya da fonksiyonları belirtirler. Örneğin, aşağıdaki bildirimlerin global düzeyde yapıldığını varsayalım:

```
static int g_a;
static void foo(void);
```

Burada *g_a* ve *foo* içsel bağlanıma sahiptir. Dolayısıyla *g_a* isimli nesneye ve *foo* fonksiyonuna yalnızca bu derleme biriminden erişilebilir. *static* global değişken bildirimlerinin nesne yaratılmasına yol açacağını vurgulayalım. Dolayısıyla yukarıdaki *g_a* bildirimi aynı zamanda bir tanımlama işlemidir. Burada bir nokta üzerinde durmak istiyoruz: *C*'de yerel bildirimlerdeki *static* belirleyicileri tamamen farklı bir anlam ifade ederler. Yerel bildirimlerdeki *static* belirleyicileri bağlanım üzerinde etkili olmamaktadır. *static* yerel nesneler blok faaliyet alanı (*block scope*) kuralına uyarlar ve *static* ömürlüdürler. Yani bu nesneler programın çalışmasıyla yaratılırlar ve akış blok dışına çıksa bile yaşamaya devam ederler. Ayrıca *C*'de yerel düzeyde *extern* bir fonksiyon bildirimi (*prototipi*) bulundurulabildiği halde *static* bir fonksiyon bildirimi bulundurulamamaktadır. Örneğin:

```
void foo(void)
{
    extern int g_a;          /* geçerli */
    static int x;            /* geçerli fakat x'in bağlanımı yoktur */
    extern int y;            /* geçerli, y dışsal bağlanıma sahip */
    extern void bar(void);   /* geçerli, extern yazılmayabilirdi */
    static void tar(void);   /* geçersiz, yerel düzeyde static fonksiyon
                                bildirimi yapılamaz */
    /* ... */
}
```

extern bildiriminin bazı ayrıntıları vardır. Bir değişken *extern* olarak bildirilmişse bağlanım için değişkenin daha önceki bildirimlerine bakılır. Eğer değişkenin daha önceden (yani daha yukarıda) yapılmış bir bildirimi varsa daha önceki bildirimde belirtilen bağlanım esas alınır. Eğer değişkenin daha önce yapılmış bir bildirimi yoksa ya da bağlanıma sahip olmayan bir bildirimi varsa değişken dışsal bağlanıma sahip olur. Bu anlatımlara göre, örneğin bir değişken önce *static* sonra *extern* olarak bildirilebilir. Bu durumda değişken içsel bağlanıma sahip olur:

```
static void foo(void);
extern void foo(void);     /* geçerli, bağlanım dışsal değil, içsel */
void foo(void);            /* geçerli, yukarıdakiyle eşdeğer */
```

Yukarıdaki durumun tersi tanımsız davranışa (*undefined behavior*) yol açmaktadır. Yani bir değişkenin önce *extern* sonra *static* olarak bildirilmemesi gerekir. Eğer bu biçimde bildirimler yapılırsa derleme aşamasında hata oluşmaz ancak bağlanım konusunda belirsizlik oluşur. Örneğin:

```
void foo(void);

/* ... */

static void foo(void)     /* dikkat: bağlanımın ne olduğu belli değil! */
{
    /* ... */
}
```

Nihayet *C*'de global fonksiyon ve global nesne bildirimlerinin dışındaki bildirimlerdeki değişkenlerin (yapı, *typedef* vs. gibi), fonksiyonların parametre değişkenlerinin ve *extern* belirleyicisi olmadan bildirilmiş yerel değişkenlerin bağlanımları yoktur.

Tüm anlatılanları aşağıdaki soru ve yanıtlarla daha açıklayıcı hale getirebiliriz:

(S): Global bir nesne ya da fonksiyon önce *static* sonra *extern* belirleyicileri ile bildirilebilir mi?
(Y): Bildirilebilir, bu durumda değişkenin bağlanımı içsel olur.

(S): Global bir değişken önce *extern* sonra *static* olarak bildirilebilir mi?
(Y): Bildirilmesi doğru olmaz. Bildirilirse programın sağlıklı çalışması garanti edilemez. *Buradaki uygunsuzluk C'*de tanımsız davranış (*undefined behavior*) terimiyle ifade edilmektedir.

(S): İçsel bağlanıma sahip olmasını istediğim bir fonksiyonun prototipini yukarıya, tanımlamasını aşağıya yerleştirmek istiyorum. Yalnızca tanımlamada *static* belirleyicisini kullanmam yeterli olur mu? Yani aşağıdaki durum geçerli midir?

```
void foo(void);

/* ... */

static void foo(void)
{
    /* ... */
}
```

(Y): Eğer prototipte *static* belirleyicisini kullanmazsanız sanki *extern* kullanmışsınız gibi etki oluşur. Bir fonksiyonun önce *static* sonra *extern* belirleyicileri ile bildirilmesi tanımsız davranışa yol açmaktadır.

(S): Peki *static* fonksiyonun prototipinde *static* belirleyicisini kullandığımı varsayalım. Aşağıdaki tanımlamasında *static* belirleyicisini kullanmak zorunda mıyım? Yani aşağıdaki durum geçerli midir?

```
static void foo(void);

/* ... */

void foo(void)
{
    /* ... */
}
```

(Y): Tanımlamada *static* belirleyicisini kullanmadığınız için *extern* kullanmışsınız gibi gibi etki oluşur. Bu durumda fonksiyonun daha önceden bir bildiriminin olup olmadığına bakılacaktır. Fonksiyon daha önce içsel bağlanıma sahip olarak bildirildiği için bu bağlanım esas alınır. Yani yukarıdaki örnek geçerlidir. Tabi okunabilirlik bakımından fonksiyonun tanımlamasında da *static* belirleyicisinin kullanılması tavsiye edilir.

(S): Global bir nesne ya da fonksiyon tüm derleme birimlerinde *extern* olarak bildirilirse sorun oluşur mu?
(Y): Her derleme birimi diğerlerinden bağımsız olarak ayrı ayrı derleme işlemine sokulmaktadır. Örneğin programınız *"a.c"*, *"b.c"* ve *"c.c"* dosyalarından oluşmuş olsun. Bu dosyaların her biri bir derleme birimidir. Bu dosyalar bağımsız olarak derlenir ve oluşturulan amaç dosyalar (*object files*) bağlayıcı (*linker*) denilen programla birleştirilir. Eğer bir değişkeni tüm derleme birimlerinde *extern* olarak bildirirseniz fakat onun için bir tanımlama bulundurmazsanız programınızı oluşturan her derleme birimi sorunsuz olarak derlenir. Fakat bağlama (*link*) aşamasında sorun oluşur. Dışsal bağlama özelliğine sahip bir nesne ya da fonksiyonun toplamda tek bir tanımlamasının bulunması gerekir. Bu tanımlamanın da hangi derleme biriminde olacağının bir önemi yoktur. *extern* belirleyicisinin bir tanımlamaya yol açmadığını anımsayınız.

---

[1]: Bir programı oluşturan ve bağımsız olarak derlenebilen her bir kaynak dosyaya derleme birimi (*translation unit*) denilmektedir. Biz bir programı birden fazla parçaya bölüp o parçaları bağımsız olarak derleyebiliriz. Sonra da bu parçaları bağlayıcıyla (*linker*) birleştirerek çalıştırılabilen (*executable*) dosyayı elde edebiliriz.

[2]: Programlama dillerinde genel olarak bildirim (*declaration*) ve tanımlama (*definition*) terimleri farklı anlamlarda kullanılmaktadır. Bir değişkenin (*identifier*) kullanılmadan önce derleyiciye tanıtılması eylemine bildirim (*declaration*) denir. Nesne yaratan, yani bellekte yer ayrılmasına yol açan bildirimlere ise tanımlama (*definition*) denilmektedir. Her tanımlama bir bildirimdir fakat her bildirim bir tanımlama değildir. *C* ve *C++* standartlarında bildirim ve tanımlama arasındaki farklılık nesne yaratımından ziyade birden fazla kez yinelendiğinde bir soruna yol açıp açmama temelinde ele alınmıştır. Aynı değişkenin birden fazla tanımlaması yapılamaz fakat bildirimi yapılabilir. C standartlarında açıkça hangi bildirimlerin tanımlama anlamına geldiği ele alınmıştır.

Kaynaklar

1. Aslan. K. *A'dan Z'ye C Kılavuzu*. (1997). İstanbul: Pusula Yayıncılık

2. ANSI/ISO 9899:1990 (1990). *American National Standard for Programming Languages - C,* New York: American National Standard Institute.

3. ISO/IEC 9899:1990 (1999). *International Standard - Programming Languages - C, Second Edition.*

4. ISO/IEC 9899:1990 (2011). *International Standard - Programming Languages - C, Second Edition.*

5. Kernighan B. W., Ritchie D.M. (1988). *The C Programming Language, Second Edition*. New Jersey: Prentice Hall.