Programlama Dillerindeki Kuralların Gerekçeleri

26-Mayıs-2009

Doğal dillerdeki kurallar matematiksel anlamda kesin değildir, bunların istisnaları çoktur. Zaten bizi herhangi bir şey öğrenirken bezdiren şeylerden biridir istisnalar. Örneğin *İngilizce*'de daha yüksek demek için *high* sıfatı *higher* haline, *tall* sıfatı *taller* haline geliyor. Pek çok sıfatın bu biçimde "*er*" eki aldığını görüyoruz. Fakat bu kesin bir kural mı? Hayır, bunun pek çok istisnası var. Örneğin, (her nedense) kötü *bad* olduğu halde daha kötü için *worse* kullanılıyor. (Ana dil gibisi var mı? Hepimiz kırmızı için kıpkırmızı, mavi için masmavi denildiğini biliriz. Bunun bir kuralının olup olmadığını sorgulamamışızdır. Zaten umurumuzda da değildir. Kimse kırmızı için maskırmızı demez.)

Fakat programlama dillerinin kesin kuralları ve bu kuralların da çeşitli gerekçeleri vardır. (Örneğin "*case* ifadeleri sabit ifadesi olmak zorundadır" kuralının gerekçeleri vardır ve ben bunu geçen haftalarda açıklamıştım.) Çünkü programlama dilleri doğal diller gibi uzun bir süre içerisinde toplumsal bir yaşantıyla ortaya çıkmıyor. Bunlar mantık temelinde bilinçli olarak tasarlanıyorlar. Herşeyden önemlisi programlama dillerinin sentaks yapıları matematiksel olarak açıklanabiliyor. Bu nedenle bu tür *dillere biçimsel diller (formal languages)* denilmekte

Programlama dillerindeki bazı kuralların bazı gerekçeleri o dili kullanan herkes tarafından tahmin edilebilir. Fakat bazı gerekçelerin anlaşılması ve kavranması yüksek bir bilinç düzeyini gerektirir. Neden mi? Çünkü bazı kuralların gerekçelerini anlayabilmek için başka şeylerin bilinmesi gerekir de ondan. Örneğin, pek çok programlama dilinde *goto* ile başka bir fonksiyona atlama yapılamaz. Bu kuralın gerekçesini s*tack* mekanizmasını bilmeyen bir kişiye nasıl açıklayabiliriz?

Bazı programlama dillerinde kuralların gerekçeleri resmi dökümanlar halinde açıklanmaktadır. Örneğin *C90* ve *C99* için özel *hazırlanmış gerekçeler (rationale)* eki resmi bir açıklama niteliğindedir. Bazı programlama dillerinde ise bazı gerekçelerin standartların kendi içerisinde doğrudan açıklandığını görürüz. Örneğin, *C#*'ta bazı kuralların gerekçeleri bu biçimde standartların içerisinde belirtilmiş durumda. Peki genel olarak bir kuralın gerekçesini nasıl öğrenebiliriz? Öncelikle eğer varsa resmi dökümanlara bakmanızı tavsiye ederim. Eğer böyle bir döküman yoksa bu durumda tasarımcıların hazırladıkları kitaplar, makaleler ya da onlarla yapılan röportajlar ve söyleşiler iyi bir kaynak olabilirler. Örneğin, *Stroustrup*'un *"The Annotated C++ Reference Manual"* isimli kitabı -eski olmasına karşın- *C++* için iyi bir gerekçe açıklama dökümanı olarak kullanılabilir. Ya da örneğin *C++/CLI* için *"A Design Rationale for C++/CLI"* makalesini kullanabilirsiniz. *USENET* haber grupları, tartışma grupları, *blog*'lar yine kaynak oluşturabilirler. Zaten görüyorsunuz, ben de kendi *blog*'umda arada sırada bazı gerekçeleri açıklamaya çalışıyorum.

Peki ya tasarımcılar tarafından oluşturulan kurallar ve bunların gerekçeleri her zaman bizi ikna edebiliyor mu? Yanıt hayır! Bu bağlamda ben pek çok dildeki pek çok kuralı gerekçeleriyle birlikte eleştiriyorum. Örneğin, *C#*'taki *switch* deyiminde aşağıya *düşme kuralının (fall through)* kaldırılarak bunun dolaylı bir biçimde *goto case* ve *goto default* deyimleriyle karşılanmasını gereksiz ve saçma buluyorum. Bu konuda *Hejlsberg*'in belirttiği gerekçeler de beni tatmin etmiyor. Tabi gerekçeleri gözden

geçirirken o programlama dilinin seviyesini, kullanım alanını ve programlama modelini göz önüne almak zorunda olduğumuzun farkındayım. Fakat siz de bazen "neden şöyle değil de böyle" diye merak ettiğiniz kuralların pekala "şöyle" de olabileceğini gözardı etmeyin. Çünkü bazı durumlarda tasarımcının elinde birbirine yakın birden fazla seçenek vardır. Bu seçenekler neredeyse eşdeğer işlevselliktedir. Tasarımcı da bunlardan birini seçmiştir.

Programlama dillerindeki bazı kurallar da zamanla değiştirilebiliyor. Buna tipik bir örnek olarak *C++*'taki deyimlerin parantezleri içerisinde bildirilen değişkenlerin faaliyet alanı verilebilir:

```
for (int i = 0; i < 100; ++i) {
    //...
}
```

i değişkeni nerede bildirilmiştir? *Stroustrup* önce bunun *for* döngüsünün yerleştirildiği blokta bildirilmiş olması gerektiğini düşünmüştür. Fakat bu kural yeterince işlevsel değildir. Yukarıdaki döngüyü biraz aşağıya kopyalarsanız hata oluşur. Bu kural daha sonra (standartlarla birlikte) değiştirilmiştir. Bugün artık *C/C++* kökeninden gelen pek çok dilde bu değişkenlerin faaliyet alanları döngü bloğu ile sınırlıdır.

Programlama dillerinin tasarımında ve standartlarında böcek olmuyor mu? Tabi oluyor. Fakat bunları zamanla düzeltiyorlar. Bugün *Internet* sayesinde çeşitli ortamlarda pek çok yeni fikir tartışılabiliyor ve eski fikirler eleştirilebiliyor. Tabi geçmişe doğru uyumun bozulması bazen tüm çabalara rağmen engellenemiyor...