

C - Storage Classes

22/07/2008 15:12 by Coder

Storage Classes 7.1 Scope ve Duration mantığı Degiskenler program içinde scope ve duration özelliklerine sahiptirler. Scope, bir degiskenin tanımlı olduğu bölge iken, duration degiskenin hafızada tutulma süresidir. Bir degiskenin scope'u ve duration'una birden, o degiskenin storage class'i yani 'depolama sinifi' adi verilir. 7.2 Fixed (Sabit) ve Automatic (Otomatik) Duration Fixed duration'a sahip degiskenlerin hafızada tutulacakları yer önceden ayrılmıştır; yani sabittir. Automatic duration'a sahip degiskenlerin tutulacakları hafıza alanları ise programın çalışması sırasında ayrılır. Fixed duration'a sahip degiskenin alanı programın başlangıcında ayrılır ve bütün program boyunca degismez. Automatic duration'a sahip bir degiskenin hafıza alanı ise, o degiskenin tanımlandığı bölgeye girildiğinde ayrılır ve bu kod parçasına her girildiğinde bu alanın yeri degisir. Sabit degisken tanımlamak için kullanılan keyword 'static' tir. Bu keyword kullanılarak tanımlanan degiskenler sabittir. Otomatik degisken tanımlamak için ise auto keyword'ü seçilmiştir. Fakat tanımlanan bütün degiskenler otomatik olarak değerlendirildiklerinden bu keyword kullanılmaz. 7.2.1 Degiskenlere ilk deger verme Sabit degiskenler program içerisinde sadece bir kez ilk deger alırken, otomatik degiskenlerin degerleri tanımlandıkları kod parçasına her giriste yenilir. Örneğin; void increment() { int j=1; static int k=1; j++; k++; printf("j: %d k: %d\n", j, k); } main() { increment(); increment(); increment(); } Bu program çalıştırıldığında k sadece 1 kez ilk deger alır ve ondan sonra k üzerinde yapılan bütün işlemler mevcut degeri üzerinden gerçekleşir. Bu programın ekran çıktısı; j: 2 k: 2 j: 2 k: 3 j: 2 k: 4 increment() fonksiyonu her çağırıldığında j 1 degerini alır ve j++ ile degeri 2 olur. Fakat k 1 kez 1 degerini aldıktan sonra, increment() her çağırıldığında bu deger 1 artırılır. Sabit ve otomatik duration'a sahip degiskenler arasındaki bir diğer önemli fark da, otomatik degiskenlere ilk deger verilmediğinde ayrılan adresteki degerleri almalarına karşın, sabit degiskenlerin ilk degeri her zaman 0'dır. Az önceki örnekte j'ye ve k'ye ilk deger atama işlemini kaldırırsak oluşacak ekran görüntüsü; j: 243653 k: 1 j: 243653 k: 2 j: 243653 k: 3 gibi birşey olacaktır. 7.3 Scope Bir degiskenin scope'u, o degiskenin tanımlı olduğu aralığa, yani o degiskenin erişebileceği aralığa denir. C'de dört ayrı scope (bölge) vardır: Program scope: Programın bütünü oluşturan farklı kod parçaları tarafından degiskenlere erişilebilir. Bu tip degiskenlere 'global degisken' adi verilir. File scope: Tanımlanan degiskenin sadece tanımlandığı kod dosyasında aktif olduğunu ifade eder. Function scope: Degiskenin tanımlandığı fonksiyon içinde geçerli olduğunu belirtir. Block scope: Degiskenin tanımlandığı yerden, tanımlanmış olduğu bloğun sonuna kadar geçerli olduğunu belirtir. Bloklar {} ile belirtilir. Bu 4 bölge hiyerarşik olarak gösterilirse; Genelde degiskenlerin scope'larını belirleyen tanımlandıkları yerlerdir. Örneğin; int i; // Program scope static int j; // File scope func(k) // Program scope int k; // Block scope { int m; // Block scope start: // Function scope …… Degiskenlerin farklı bölgelerde tanımlı olması özelliğini kullanarak, tanım bölgeleri (Scopeleri) birbirinden farklı olmak koşuluyla aynı isimde birden fazla degisken kullanmak mümkündür. Örneğin; func1() { int j; …… } func2() { int j; …… } Aynı isimde ve tanım bölgeleri kesilen degiskenler kullanmak da mümkündür. Bu gibi durumlarda tanım aralığı daha küçük olan degisken öncelik kazanır. Örneğin; int j=10; main() { int j; for (j=0;j Bu kod parçasında global tanımlanan j degiskeni, 10 degerini korurken, main() içinde block scope'una sahip olan j'nin degeri kullanılır. 7.3.1 Block Scope Block scope'una sahip bir degiskenin block dışından müdahale edilemez. Bu da program yazma işlemini kolaylaştırır. Degiskenlerin tanım bölgelerini daraltarak daha anlaşılabilir ve düzenlenebilir kod yazmak mümkündür. Bu sayede kullanılan degiskenin degerinin degismesine yol açacak yan etkilerden de kurtulmuş olunur. Block scope kullanmak degisken isimlerinin çakışması derdini de ortadan kaldırır. while, for, if gibi kontrol ifadelerinde de kod blokları kullanıldığından, bu bloklar içinde de degisken tanımlamak mümkündür. Bu daha çok kontrol - debugging amaçlı kod yazma işleminde faydalı olur. Çok karmaşık kod parçalarında bu yöntem ile degiskenlerin çakışması da engellenir. Örneğin; foo() { int ar[20]; int j; // debugging kodu { int j; //bir üste bloktaki j ile çakışmaz for (j=0;j 7.3.2 Function scope Function scope'a sahip olan tek eleman goto komutu ile birlikte kullanılan label'lerdir. Label'ler fonksiyonun başından sonuna kadar tanımlıdır. 7.3.3 File ve Program Scope Bir degiskenin file (dosya) scope'una sahip olması, onun o dosyada yer alan bütün fonksiyonlar tarafından kullanılabilmesi anlamına gelir. Bir degiskenin file scope'una kazandırmak için, herhangi bir fonksiyonun dışında

static keyword'u ile tanımlanması gerekir. Örneğin; Program scope'a sahip değişkenler, yani global değişkenler, programı oluşturan bütün fonksiyonlar ve kod tarafından erişilebilir değişkenlerdir. Global değişken tanımlamak için, herhangi bir fonksiyonun içinde olmayacak şekilde ve static keyword'ünü kullanmadan değişkeni tanımlamak yeterlidir. Örneğin;

```
static int filesv; // File scope  
int progs; // Program scope  foo() { &hellip;
```

Kaynak: SELAM.tk