BLG339
PROGRAMLAMA DİLLERİ KAVRAMI

Hafta 4 Ders 2

Yrd. Doç. Dr. Melike Şah Direkoğlu

Alındığı kaynak:
Addison-Wesley's Programming Language Concepts slaytları ve Prof. Dr. Tuğrul Yılmaz' ın ders notlarından faydalanarak hazırlanmıştır.

İsimler ve Kapsam

- Tanım: Bir değişkenin kapsamı (scope) değişkenin görülebilir olduğu komutların alanıdır. Görülebilir olduğu alan, bir komut içinde belirlenen değerle kullanılabildiği alandır.
- Tanım: Lokal değişkenler, bir program biriminde kullanılan ve orada tanımlanmış değişkenlerdir.
- Tanım: Lokal olmayan değişkenler, bir program biriminde görülebilir olan ancak orada tanımlanmamış değişkenlerdir.
- Kapsam kuralları, isimlere yapılan referansların değişkenlerle nasıl bağlanacağının kurallarını belirler.

Statik Kapsam (Static Scope)

- Bir isim referansını değişkene bağlayabilmek için isim tanımlanmış olmalıdır.
- Arama işlemi: lokalden başlayarak ve her seferinde kapsamı genişleterek, verilen ismin tanımını arama. Bu durumda kapsam en içteki alt programdan onu çevreleyen üst alt programlara doğrudur.
- Bazı diller iç içe alt programları desteklerken (Ada, JavaScript, PHP),
- Bazı diller iç içe alt programları desteklemez (C tabanlı diller gibi).
- İç içe alt programları desteklemeyen dillerde bile blok içleri ayrı kapsama alanlarıdır.

Blok Kavramı

- Program birimleri içerisinde statik kapsam yaratma yöntemi – ALGOL 60 ile başlar
- Örnekler:
- C and C++: **for (...)**
  {
  int index;
  ...
  }

- Ada: **declare LCL : FLOAT;**
  begin
  ...
  end

Kapsam

- Lokalde tanımlanmış aynı isimli değişken, dışarda tanımlanmış değişkene erişimi keser:

```
void sub() {
    int count;
    …
      while ( … ) {
      int count=1;
      count ++;
      …
      }
    }
```

- C++ ve Ada bu tip erişilmez verilere kapsamı belirterek erişim imkanı sağlar.
  - Ada: **unit.name**
  - C++: **class_name::name**

Statik Kapsam Örnekler

- C++ değişken tanımlarının fonksiyon içinde herhangi bir yerde yapılmasına izin verir. Fonksiyonun içinde ama bir blok içinde olmayan tanımlar, fonksiyon içinde tanımlandığı noktadan fonksiyonun sonuna kadar tanımlanmış sayılırlar.
- C'de benzer tanımların fonksiyon başında yapılması zorunludur.
- C++, Java ve C# "class"ları içinde tanımlanan değişkenler farklılıklar gösterir:
  - Eğer herhangi bir metodun içinde tanımlanmadıysa, bütün class içinde tanımlıdır. "public"se dışardan da erişilebilir.
  - Bir metot içinde tanımlandıysa, tanımlandığı blokdaki değerini kullanır.
  - C#, C++ tipi göstericileri destekler. Ancak bunlar güvenliği bozduklarından bunları kullanan 'metot'ların 'unsafe' olarak tanımlanması zorunludur.

Kapsam

- Statik kapsamın değerlendirilmesi
- Örnek: Bütün kapsamlar MAIN program ve alt programlarca belirlenir.

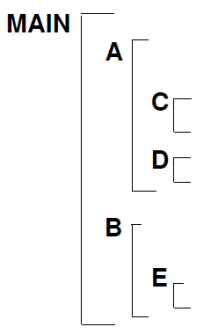
  MAIN A ve B yi çağırır

  C ve D'yi A çağırır

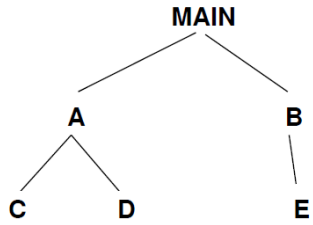
  A ve E'yi B çağırır

Statik Kapsam Örneği

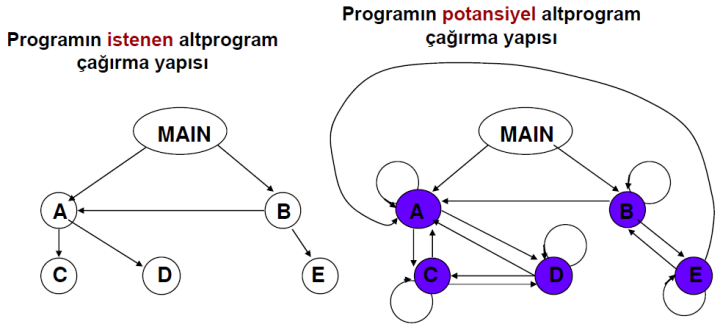
Programın yapısı



Programın ağaç yapısı

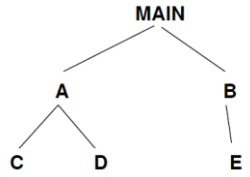


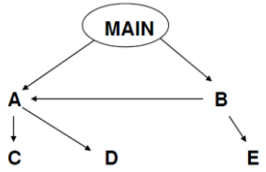
Statik Kapsam Örneği



Statik Kapsam Örneği

- Varsayalım D, B'nin içindeki veriye ulaşmalı.
- Olası çözümler:
  - D'yi B'nin altına koy (Fakat artık C, D'yi çağıramaz, D, A'nın değişkenlerine erişemez).
  - B'den D'nin gereksinim duyduğu veriyi MAIN'e koy (bütün alt programlar erişebilir)
- **Sonuç: statik kapsam global değişkenlere neden olur.**





Dinamik Kapsam (Dynamic Scope)

- Program birimlerinin çağırma sırasına dayanır; onların programdaki yerleşme şekillerine değil.
- Değişkenlere erişim altprogramların herhangi bir andaki çağrı zincirine bağlıdır.
- APL, SNOBOL4, Perl ve bazı Lisp versiyonları dinamik kapsam kullanır. Perl ve Common Lisp her iki kapsamı da kullanabilirler.

Dinamik Kapsama Örnek

```
function big() {
  function sub1() {
    var x = 7;
  }
  function sub2() {
    var y = x;
    var z = 3;
  }
  var x = 3;
}
```

- Sub2() deki x in kapsamı dinamiktir. Anlamı: x'in değeri derleme sırasında belirlenemez.
  - Program birimlerinin çağrılış sırasına göre x in değeri de değişecektir.
- Dinamik kapsamanın değerlendirmesi:
  - avantaj: kolaylık;
  - dezavantaj: zor okunabilirlik.

Dinamik Kapsam Değerlendirmesi

- Bir altprogram içerisinde tanımlanmamış değişken, programın sürecine farklı altprogramlardaki farklı tanımlara gönderme yapıyor olabilir.
- Altprogramlardaki değişkenleri başka altprogramların beklenmedik değiştirmelerinden korumak çok zor. Güvenilirlik çok düşüyor.
- Yerel olmayan değişkenlerin kullanım sırasında tip kontrolünü yapmak zor.
- Dinamik kapsamlı bir programı okumak pratikte çok zor. Her türlü dinamik kapsam öngörülemez.
- Yerel olmayan değişkenlere erişim çok fazla zaman aldığından, program yavaşlıyor.

Kapsam ve Yaşam Süresi

- Kapsam ve yaşam süresi bir birleriyle ilgili ancak farklı kavramlardır.
- Bir Java metodunun içinde tanımlanmış değişken, metot içinde geçerlidir, yaşam süresi de metot çalıştığı sürecedir.
- C ve C++'da altprogram içinde **static değişkenleri düşünün.** Kapsamı sadece altprogramdır, fakat ana program çalıştığı sürece korunur; yaşam süresi programın yaşam süresi kadardır.
- Aşağıdaki örneğe bakarsak, 'sum' değişkeninin kapsamı 'compute' fonksiyonu ile sınırlıysa da, 'printheader' çalışırken de yaşamaya devam eder.

```
void printheader() {
  ...
 }  /* end of printheader */
void compute() {
  int sum;
  ...
  printheader();
 }  /* end of compute */
```