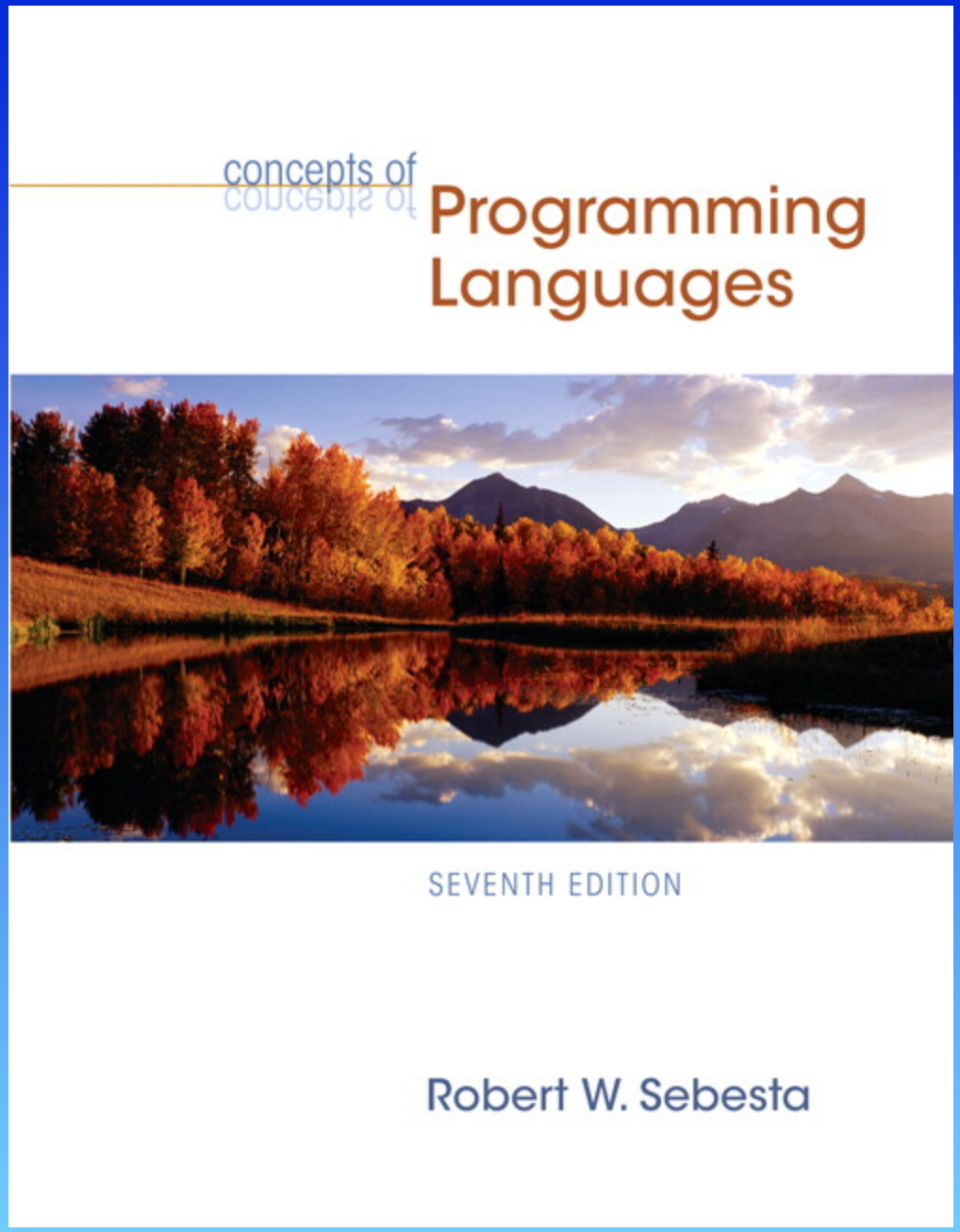
Bölüm 5

Adlar(Names), İlişkilendirmeler(Bindings), Tip Kontrolü(Type Checking), ve Kapsamlar(Scopes)



ISBN 0-321-33025-0

Bölüm 5 Konular

1. Giriş
2. Adlar(names)
3. Değişkenler(variables)
4. Bağlama(binding)Kavramı
5. Tip Kontrolü(Type Checking)
6. Kesin Tiplendirme(Strong Typing)
7. Tip Uyumluluğu(Type Compatibility)

Bölüm 5 Konular (devamı)

8. Kapsam(Scope) ve Ömür(Lifetime)
9. Referans(Kaynak Gösterme) Platformları(Referencing Environments)
10. Adlandırılmış sabitler(Named Constants)

5.1 Giriş

- Zorunlu Diller(Imperative Languages), von Neumann mimarisinin soyutlamalarıdır (abstractions)
  - Bellek(Memory)
  - İşlemci(Processor)
- Özelliklerle(attributes) tanımlanan değişkenler(variable)
  - Tip(Type): tasarlamak için, kapsam(Scope), ömür(Lifetime), tip kontrolü(type checking), başlatma(initialization), ve tip uyumluluğu(type compatibility) üzerinde düşünülmelidir.

5.2 Adlar(names)

- Adlar(names) için tasarım sorunları:
  - Maksimum uzunluk?
  - Bağlaç(connector) karakterleri kullanılabilir mi?
  - Adların(names) büyük-küçük harfe duyarlılığı(case sensitive) var mıdır?
  - Özel sözcükler(special words) ayrılmış sözcükler (reserved words) mi veya anahtar sözcükler midir(keywords)?

5.2 Adlar(names) (devamı)

- Uzunluk(Length)
  - Eğer çok kısa ise, anlaşılmaz
  - Dil örnekleri:
    - FORTRAN I: maksimum 6
    - COBOL: maksimum 30
    - FORTRAN 90 ve ANSI C: maksimum 31
    - Ada ve Java: limit yoktur, ve hepsi anlamlıdır(significant)
    - C++: limit yoktur, fakat konabilir

5.2 Adlar(names) (devamı)

- Bağlaçlar(Connectors)
  - Pascal, Modula-2 ve FORTRAN 77 kabul etmez
  - Diğerleri eder

5.2 Adlar(names) (devamı)

- Büyük küçük harf duyarlılığı (Case sensitivity)
  - Dezavantaj: okunabilirlik(readability) (benzer görünen adlar(names) farklıdır)
    - C++ ve Java'da daha kötüdür çünkü önceden tanımlanmış(predefined) adlar(names) karışık büyük-küçüklüktedir (örn. **`IndexOutOfBoundsException`**)
  - C, C++, ve Java adları(names) büyük küçük harfe duyarlıdır(case sensitive)
  - Diğer dillerdeki adlar(names) değildir

5.2 Adlar(names) (devamı)

- Özel Sözcükler(Special words)
  - Okunabilirliğe yardımcı olmak için; ifade yantümcelerini(statement clauses) sınırlamak ve ayırmak için kullanılır
  - Tanım: Bir anahtar sözcük(keyword) yalnızca belirli bir bağlamlarda(kontekstler)(contexts) özel olan sözcüktür(word) örn. Fortran'da:

    Real VarName (*Real arkasından bir ad(name) gelen bir veri tipidir(data type), bu yüzden Real bir anahtar sözcüktür (keyword))*

    Real = 3.4 (*Real bir değişkendir(variable))*
  - Dezavantaj: zayıf okunabilirlik
  - Tanım: Bir ayrılmış sözcük(reserved word) kullanıcı-tanımlı ad (a user-defined name) olarak kullanılamayan bir özel sözcüktür(word)

5.3 Değişkenler(variable)

- Bir değişken(variable) bir bellek hücresinin (memory cell) soyutlanmasıdır(abstraction)
- Değişkenler(variables), özelliklerin(attributes) bir altılısı olarak karakterize edilebilir:

  (ad(name), adres(address), değer(value), tip(type), ömür(Lifetime) ve Kapsam(Scope))
- Ad(Name) – bütün değişkenler(variables) onlara sahip değildir (adsız(anonim) (anonymous))

5.3 Değişkenler(variable) (devamı)

- Adres(Address) – ilgili olduğu bellek adresi(memory address) (*l*–value de denir)
  - Bir değişken(variable) çalışma süresi boyunca farklı zamanlarda farklı adreslere sahip olabilir
  - Bir değişken(variable) bir program içerisinde farklı yerlerde faklı adreslere sahip olabilir
  - Eğer iki değişken adı(variable names) aynı bellek konumuna erişmek için kullanılabiliyorsa, bunlara takma ad(diğer ad)(alias) adı verilir
  - Takma adlar(Aliases) okunabilirlik açısından zaralıdır (program okuyucuları hepsini hatırlamak zorundadır)

5.3 Değişkenler(variable) (devamı)

- Takma adlar(aliases) nasıl oluşturulabilir:
  - İşaretçiler(Pointers), referans değişkenleri(reference variables), C ve C++ bileşimleri(unions), (ve geçiş parametrelerle- Bölüm 9 da bahsedilecek)
  - Takma adlar(aliases) için orijinal gerekçelerin bazıları artık geçerli değildir; örn. FORTRAN'da belleğin yeniden kullanımı
  - Bunlar dinamik ayırma(dynamic allocation) ile değiştirilir

5.3 Değişkenler(variable) (devamı)

- Tip(Type) – değişken(variable) değerlerinin aralığını(range) ve o tipin(type) değerleri için tanımlanan işlemler kümesini belirler; kayan-nokta(floating point) olduğu durumda, tip(type) aynı zamanda duyarlılığı(precision) da belirler
- Değer(Value) – değişken(variable) ile ilişkilendirilmiş olan konumun içeriği
- Soyut bellek hücresi(Abstract memory cell) – değişken(variable) ile ilişkilendirilmiş olan fiziksel hücre veya hücreler koleksiyonu

5.4 Bağlama(binding)kavramı

- Bir değişkenin(variable) *l*–değeri(*l*–value) onun adresidir(address)
- Bir değişkenin(variable) *r*–değeri(*r*–value) onun değeridir(value)
- Tanım: Bağlama(binding) bir ilişkilendirmedir, bir özellik(attribute) ve bir varlık(entity) arasında, veya bir işlem(operation) ve bir sembol(symbol) arasındaki gibi.
- Tanım: Bağlama süresi(binding time) bir bağlamanın(binding) meydana geldiği süredir

5.4 Bağlama(binding)kavramı(devamı)

- Olası bağlama süreleri(binding times):
  - Dil tasarım süresi(design time)--örn., operatör sembollerini(operator symbols) işlemlere(operations) bağlama
  - Dil implementasyon süresi(implementation time)--örn., kayan nokta tipini(floating point type) gösterime(representation) bağlama
  - Derleme süresi(Compile time)--örn., bir değişkeni(variable) C veya Java'daki bir tipe(type) bağlama
  - Yükleme süresi(Load time)--örn., bir FORTRAN 77 değişkenini(variable) bir bellek hücresine(memory cell) bağlama (veya bir C `static` değişkenine(variable))
  - Yürütme süresi(Runtime)--örn., bir statik olmayan(nonstatic) lokal değişkeni(local variable) bir bellek hücresine(memory cell) bağlama

5.4 Bağlama(binding)kavramı(devamı)

- Tanım: Bir bağlama(binding) eğer yürütme süresinden(run time) önce meydana geliyor ve programın çalışması boyunca değişmeden kalıyorsa statiktir(static) .
- Tanım: Bir bağlama(binding) eğer yürütme süresi(run time) sırasında meydana geliyor veya programın çalışması sırasında değişebiliyorsa dinamiktir(dynamic).

5.4 Bağlama(binding)kavramı(devamı)

- Tip bağlamaları(Type bindings)
  - Bir tip nasıl belirlenir?
  - Bağlama(binding) ne zaman meydana gelir?
  - Eğer statik ise, tip ya açık(belirtik)(explicit) veya örtük(implicit) bildirim(declaration) ile belirlenebilir

5.4 Bağlama(binding)kavramı(devamı)

- Tanım: Açık bildirim(explicit declaration) değişkenlerin(variable) tiplerini tanımlamak için kullanılan bir program ifadesidir(statement)
- Tanım: Örtük bildirim(implicit declaration) değişkenlerin(variable) tiplerini belirlemek için varsayılan bir mekanizmadır (değişkenin(variable) programdaki ilk görünüşü)
- FORTRAN, PL/I, BASIC, ve Perl örtük bildirimler(implicit declarations) sağlar
  - Avantaj: yazılabilirlik(writability)
  - Dezavantaj: güvenilirlik(reliability) (Perl'de daha az problem)

5.4 Bağlama(binding)kavramı(devamı)

- Dinamik Tip Bağlama(Dynamic Type Binding) (JavaScript ve PHP)
- Bir atama ifadesiyle(assignment statement) belirlenir. örn., JavaScript

```
list = [2, 4.33, 6, 8];
list = 17.3;
```

  - Avantaj: esneklik(flexibility) (soysal(generic) program birimleri(units))
  - Dezavantajlar:
    - Yüksek maliyet (dinamik tip kontrolü(dynamic type checking) ve yorumlama(interpretation))
    - Derleyici(compiler) ile tip hatası saptamak(Type error detection) zordur

5.4 Bağlama(binding)kavramı(devamı)

- Tip çıkarım(Type Inferencing)(ML, Miranda, ve Haskell)
  - Atama ifadesi(assignment statement) yerine, tipler(types) referansın(reference) bağlamından(context) belirlenir
- Bellek Bağlamalar(Storage bindings) & Ömür(Lifetime)
  - Ayırma(Allocation) – kullanabilir hücreler(cells) havuzundan bir hücre almak
  - Serbest Bırakma(Deallocation) – bir hücreyi(cell) havuza geri koymak
- Tanım: Bir değişkenin(variable) ömrü(Lifetime) onun belli bir bellek hücresine(memory cell) bağımlı olduğu sürece geçen zamandır

5.4 Bağlama(binding)kavramı(devamı)

- Ömürlerine(Lifetimes) göre değişkenlerin(variables) kategorileri
  - Statik(Static)—çalışma başlamadan önce bellek hücrelerine(memory cells) bağlanır ve çalışma süresince aynı bellek hücresine bağımlı kalır.

    örn. Tüm FORTRAN 77 değişkenleri(variables), C statik değişkenleri
  - Avantajlar: verimlilik(efficiency) (direk adresleme), tarih-duyarlı altprogram(history-sensitive subprogram) desteği
  - Dezavantaj: esnek (flexibility) olmaması (özyineleme(recursion) yoktur)

5.4 Bağlama(binding)kavramı(devamı)

- Ömürlerine(Lifetimes) göre değişkenlerin(variables) kategorileri
  - Yığın-dinamik(Stack-dynamic)—Bellek bağlamalar(Storage bindings) değişkenler(variables) için bildirim ifadeleri (declaration statements) incelendiği zaman oluşturulur
  - Eğer skaler(scalar) ise, adres dışındaki bütün özellikler(attributes) statik olarak bağlanmıştır

    örn. C altprogramlarındaki lokal değişkenler(local variables) ve Java metotları (methods)
  - Avantaj: özyinelemeye (recursion) izin verir; belleği korur
  - Dezavantajlar:
    - Ayırma (allocation) ve serbest bırakma(deallocation) nın getirdiği ek yük(overhead)
    - Altprogramlar(Subprograms) tarih-duyarlı olamaz (history sensitive)
    - Verimsiz referanslar (dolaylı adresleme(indirect addressing))

5.4 Bağlama(binding)kavramı(devamı)

- Ömürlerine(Lifetimes) göre değişkenlerin(variables) kategorileri
  - Açık altyığın-dinamik(Explicit heap-dynamic)— Açık(belirtik)(explicit) yönergeler(directives) tarafından ayrılır(allocated) ve serbest bırakılır(deallocated), programcı tarafından belirlenmiştir, çalışma süresi boyunca etkili olur
  - Sadece işaretçiler(pointers) veya referanslar(references) ile başvurulur

    örn. C++ 'taki dinamik nesneler (new ve delete yoluyla)

    Java daki bütün nesneler
  - Avantaj: dinamik bellek yönetimi sağlar
  - Dezavantaj: verimsiz ve güvenilmezdir

5.4 Bağlama(binding)kavramı(devamı)

- Ömürlerine(Lifetimes) göre değişkenlerin(variables) kategorileri
  - Örtük altyığın-dinamik(Implicit heap-dynamic)—atama ifadelerinin sebep olduğu ayırma(Allocation) ve serbest bırakma(deallocation)

    örn. APL 'deki bütün değişkenler(variables); Perl ve JavaScript 'deki bütün stringler ve diziler(arrays)
  - Avantaj: esneklik(flexibility)
  - Dezavantajlar:
    - verimsizdir, çünkü bütün özellikler(attributes) dinamiktir
    - Hata saptama kaybı(error detection)

5.5 Tip Kontrolü(Type checking)

- İşlenenler(Operands) ve operatörler kavramını altprogramlar(subprograms) ve atamaları(assignments) içerecek şekilde genelleştirmek
- Tip Kontrolü(Type checking) bir operatörün(operator) işlenenlerinin(operands) uyumlu tiplerde(compatible types) olmasını güvence altına alma faaliyetidir.
- Bir uyumlu tip(compatible type), ya operatör için legal olan, veya derleyicinin(compiler) ürettiği kod ile dil kuralları(rules) altında örtük(dolaylı) olarak (implicitly) legal bir tipe çevrilmesine izin verilen tiptir. Bu otomatik dönüştürmeye(conversion) zorlama(coercion) adı verilir.
- Bir tip hatası(type error), bir operatörün uygun olmayan tipteki bir işlenene(operand) uygulanmasıdır

5.5 Tip Kontrolü(Type checking) (devamı)

- Eğer bütün tip bağlamaları(type bindings) statik ise, neredeyse tüm tip kontrolü(Type checking) statik olabilir
- Eğer tip bağlamaları(type bindings) dinamik ise, tip kontrolü(Type checking) dinamik olmalıdır
- Tanım: Bir programlama dilinde eğer tip hataları(type errors) her zaman saptanıyorsa, o dil kesin/kuvvetli tiplendirilmiştir(strongly typed)

5.6 Kesin Tiplendirme(Strong Typing)

- Kesin tiplendirmenin(strong typing) avantajı: değişkenlerin(variable) yanlış kullanılmasıyla oluşan tip hatalarını engeller. Dil örnekleri:
  - FORTRAN 77 böyle değildir: parametreler, **`EQUIVALENCE`**
  - Pascal böyle değildir: variant records
  - C ve C++ değildir: parametre tip kontrolü(parameter type checking) önlenebilir; bileşimler(unions) tip kontrollü değildir(type checked)
  - Ada hemen hemen böyledir, (**`UNCHECKED CONVERSION`** kaçamak noktasıdır(loophole))

    (Java buna benzerdir)

5.6 Kesin Tiplendirme(Strong Typing) (devamı)

- Zorlama kuralları(Coercion rules) kesin tiplendirmeyi(strong typing) fazlasıya etkiler—oldukça yavaşlatabilirler (C++ 'a karşı Ada)
- Java'nın C++'ın atama zorlamalarının(assignment coercions) yalnızca yarısına sahip olmasına rağmen, onun kesin tiplendirmesi(strong typing) Ada'nınkinden çok daha az etkilidir

5.7 Tip Uyumluluğu(Type Compatibility)

- Öncelikle yapısal tiplerle(structured types) ilgileniyoruz
- Tanım: Ad tipi uyumluluğu(Name type compatibility),iki değişkenin(variable) aynı bildirimde(declaration) veya aynı tip adını(type name)kullanan bildirimlerde olması durumunda uyumlu tiplere sahip olması anlamına gelir
- Gerçekleştirilmesi kolaydır fakat çok kısıtlayıcıdır:
  - İnteger tiplerinin altaralıkları(subranges) integer tipleriyle uyumlu(compatible) değildir
  - Formal parametreler onlara karşılık gelen güncel(actual) parametreler ile aynı tipte olmalıdır (Pascal)

5.7 Tip Uyumluluğu(Type Compatibility) (devamı)

- Yapı(Structure) tipi uyumluluğu(compatibility) iki değişkenin(variable) eğer tipleri özdeş (identical) yapılara sahipse uyumlu tiplere sahip olması anlamına gelir
- Daha esnektir, fakat gerçekleştirilmesi zordur

5.7 Tip Uyumluluğu(Type Compatibility) (devamı)

- İki yapısal tipin(structured types) problemini düşünelim:
  - Eğer iki tutanak(kayıt) tipi(record types) yapısal olarak aynı ise fakat farklı alan adları(field names) kullanıyorlarsa bunlar uyumlu mudur?
  - Eğer iki dizi tipi (array types) altsimgelerinin(subscripts) farklı olması dışında aynıysa uyumlu mudur?

    (örn. [1..10] ve [0..9])
  - Eğer iki sayım tipinin(enumeration types) bileşenlerinin yazılışları farklı ise bunlar uyumlu mudur?
  - Yapısal tip(structural type) uyumluluğuyla, aynı yapıya ait farklı tipleri ayırt edemezsiniz (örn. Farklı hız birimleri, ikisi de float)

5.7 Tip Uyumluluğu(Type Compatibility) (devamı)

- Dil örnekleri:
  - Pascal: genellikle yapı(structure), fakat bazı durumlarda ad(name) kullanılır (formal parametreler)
  - C: yapı, tutanaklar(kayıtlar)(records) için hariç
  - Ada: adın(name) kısıtlanmış biçimi
    - Türetilmiş(Derived) tipler aynı yapıdaki(structure) tiplerin farklı olmasına izin verir
    - Anonim tiplerin hepsi benzersizdir(unique), hatta:

      `A, B : array (1..10) of INTEGER`:

5.8 Kapsam(Scope)

- Bir değişkenin(variable) kapsamı(Scope) onun görünür(visible) olduğu ifadelerin(statements) aralığıdır(range)
- Bir program biriminin lokal olmayan değişkenleri (nonlocal variables) görünür fakat belirtilmemiş(declared) olan değişkenlerdir
- Bir dilin kapsam kuralları(Scope rules) adlara(names) referansların(references) değişkenlerle(variables) nasıl ilişkilendirildiğini belirler

5.8 Kapsam(Scope) (devamı)

- Statik Kapsam(Static Scope)
  - Program metnine(text) dayalıdır
  - Bir değişkene(variable) bir ad referansı(name reference) bağlamak için, siz (veya derleyici(compiler)) belirtimi (declaration) bulmalısınız
  - Arama işlemi: bildirimler(declarations) aranır, ilk önce lokal olarak, sonra gittikçe daha geniş çevreleyen (enclosing) kapsamlarda(scopes), verilen ad(name) için bir tane bulunana kadar
  - Çevreleyen statik kapsamlar(Enclosing static scopes) (belirli bir kapsama(specific scope)) onun statik ataları(static ancestors) denir; en yakın statik ataya(static ancestor) statik ebeveyn(static parent) adı verilir

5.8 Kapsam(Scope) (devamı)

- Değişkenler(variables) bir birimden aynı isimli “daha yakın” ("closer") bir değişkene(variable) sahip olarak saklanabilir
- C++ ve Ada bu “gizli”("hidden") değişkenlere(variables) erişime izin verir
  - Ada’da: **`unit.name`**
  - C++’da: **`class_name::name`**

5.8 Kapsam(Scope) (devamı)

- Bloklar(Blocks)
  - Program birimleri içinde statik kapsamlar(static scopes) oluşturmanın bir metodu-- ALGOL 60'dan
  - örnekler:

C ve C++:
```
for (...)
  {
    int index;
    ...
  }
```

Ada:
```
declare LCL : FLOAT;
begin
    ...
end
```

5.8 Kapsam(Scope) (devamı)

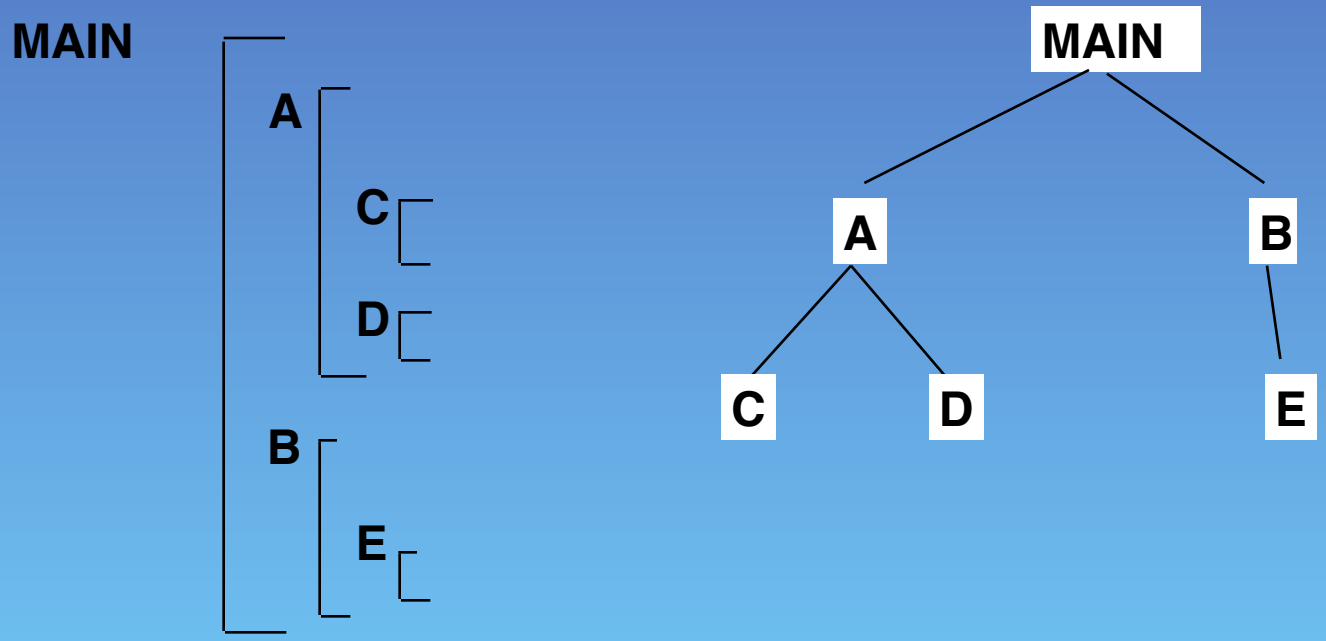
- Statik Kapsamanın(Static Scoping) değerlendirmesi
- Örneğe bakalım:

Varsayalım ki MAIN, A ve B yi çağırır

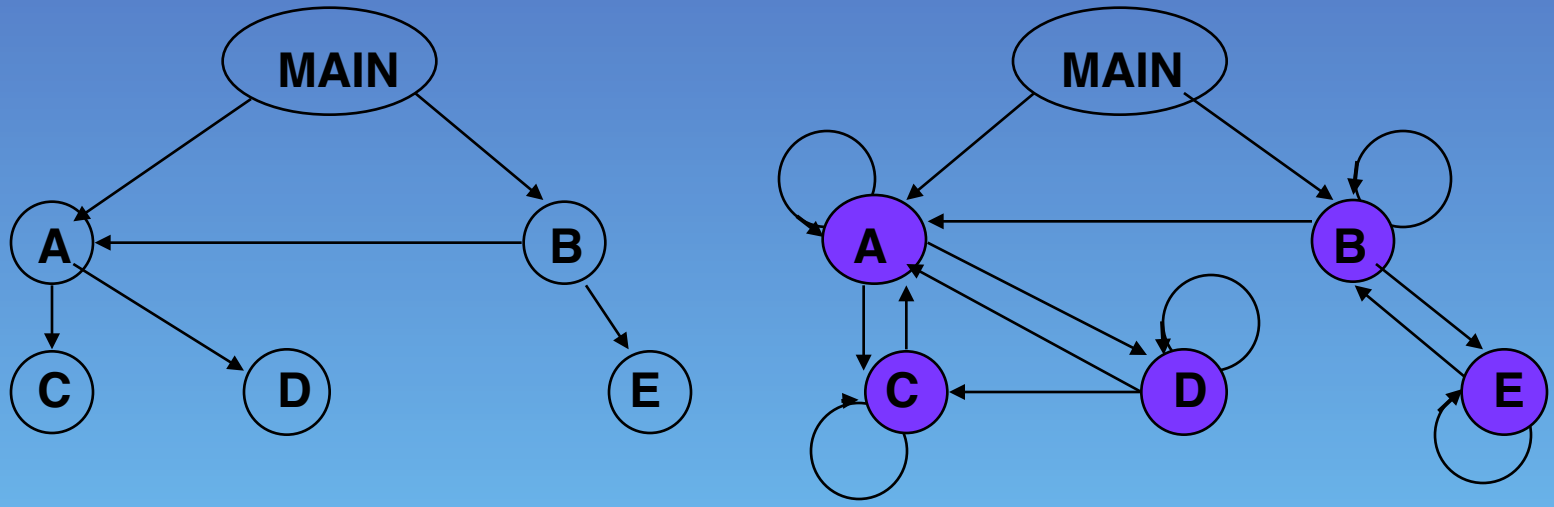
A, C ve D yi çağırır

B, A ve E yi çağırır

Statik Kapsam(Static Scope) Örneği



Statik Kapsam(Static Scope) Örneği



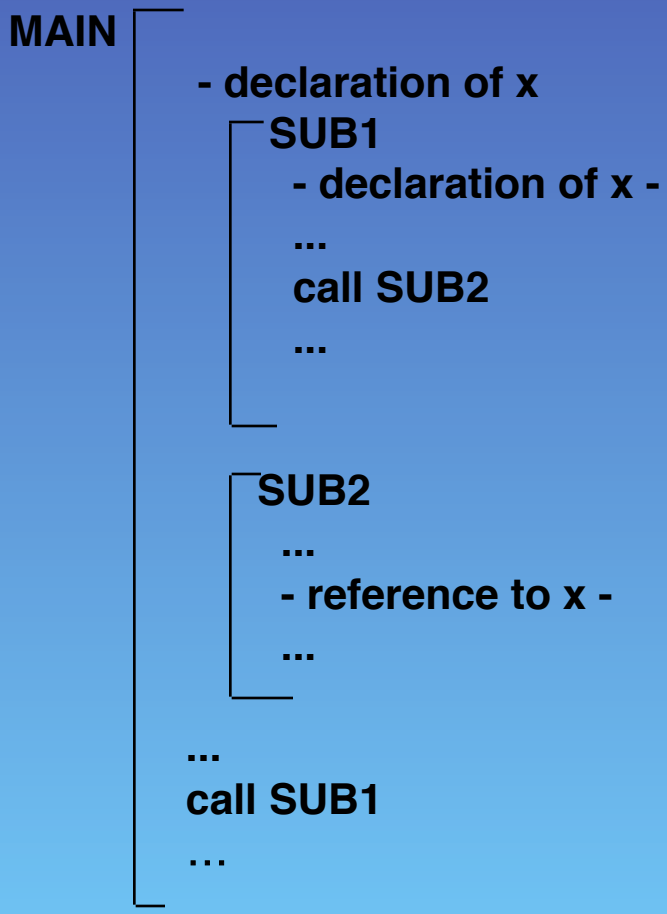
Statik Kapsam(Static Scope)

- Şartın değiştiğini varsayalım öyle ki D, B deki bazı veriye erişmek zorunda
- Çözümler:
  - D'yi B'nin içine koy (fakat o zaman C artık onu çağıramaz ve D, A'nın değişkenlerine(variables) erişemez)
  - D'nin ihtiyacı olan veriyi B'den MAIN'e taşı (fakat o zaman bütün prosedürler(procedures) onlara erişebilir)
- Prosedür erişim için aynı problem
- Sonuçta: statik kapsama(static scoping) çoğunlukla birçok globale teşvik eder

5.8 Kapsam(Scope) (devamı)

- Dinamik Kapsam(Dynamic Scope)
  - Program birimleri sıralarının çağrılmasına dayalıdır, onların metinsel düzenine(textual layout) değil, zamansala(temporal) karşı mekansal(uzaysal)(spatial))
  - Değişkenlere(variable) referanslar, bu noktaya kadar çalışmayı zorlamış altprogram çağrıları zincirinden geriye doğru arama yapma yoluyla bildirimlere(declarations) bağlıdır

Kapsam(Scope) Örneği



MAIN SUB1'i çağırır
SUB1 SUB2'yi çağırır
SUB2 x'i kullanır

Kapsam(Scope) Örneği

- Statik kapsama(Static scoping)
  - x'e referans MAIN'in x'inedir
- Dinamik Kapsama(Dynamic scoping)
  - x'e referans SUB1'in x'inedir
- Dinamik kapsamanın değerlendrilmesi:
  - Avantaj: elverişlilik
  - Dezavantaj: zayıf okunabilirlik(readability)

5.10 Referans Platformları(Referencing Environments)

- Tanım: Bir ifadenin (statement) referans platformu(referencing environment) ifadede görünen bütün adların(names) koleksiyonudur
- Bir statik kapsamlı dil(static-scoped language), lokal değişkenler(local variables) artı bütün çevreleyen (enclosing) kapsamlardaki (scopes) görünür değişkenlerin(variables) tümüdür
- Bir altprogramın(subprogram) çalıştırılması başlamışsa ama henüz bitmemişse o altprogram aktiftir
- Bir dinamik kapsamlı dilde(dynamic-scoped language), referans platformu lokal değişkenler(local variables) artı tüm aktif altprogramlardaki(subprograms) bütün görünür değişkenlerdir(variable)

5.11 Adlandırılmış sabitler (Named Constants)

- Tanım: Bir adlandırılmış sabit(named constant), sadece belleğe bağlı olduğu zaman bir değere bağlanmış olan değişkendir
- Avantajlar: okunabilirlik ve değiştirilebilirlik(modifiability)
- Programları parametrelerle ifade etmek için kullanılır
- Adlandırılmış sabitlere(named constants) değer bağlama(binding) statik (called manifest constants) veya dinamik olabilir
- Diller:
  - Pascal: sadece kalıp deyimler (literals)
  - FORTRAN 90: sabit-değerli deyimler
  - Ada, C++, ve Java: herhangi bir tipteki deyimler

Değişken başlatma (variable initialization)

- Tanım: Bir değişkeni(variable) belleğe bağlı olduğu sırada bir değere(value) bağlamaya (binding) başlatma(initialization) denir
- Başlatma(Initialization) genellikle bildirim ifadesinde(declaration statement) yapılır

örn., Java

```
int sum = 0;
```

Özet

- Büyük küçük harf duyarlılığı(Case sensitivity) ve adların(names) özel sözcüklerle(special words) ilişkisi adların(names) tasarım sorunlarını ifade eder
- Değişkenler(variables) altıkatlı ile karakterize edilir: ad(name), adres(address), değer(value), tip(type), ömür(Lifetime), kapsam(Scope)
- Bağlama(binding) özelliklerle(attributes) program varlıklarının(entities) birleştirilmesidir
- Skaler(Sayısal) değişkenler(Scalar variables) şu şekilde sınıflandırılır: statik(static), yığın-dinamik(stack dynamic), açık-altyığın dinamik(explicit heap dynamic), örtük altyığın-dinamik(implicit heap dynamic)
- Kesin tiplendirme(Strong typing) bütün tip hatalarını saptamak anlamına gelir