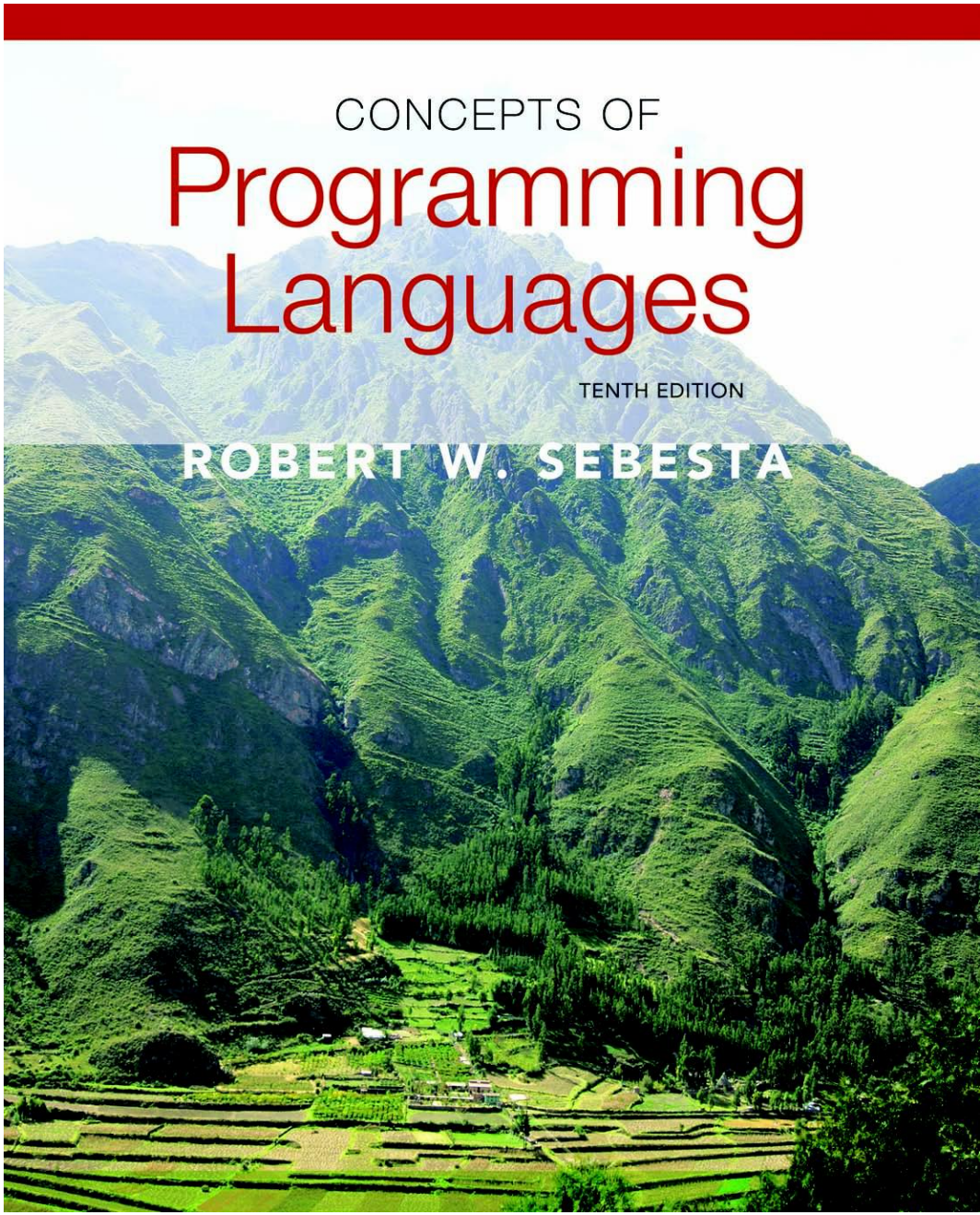
9.bölüm

Alt Programlar



9.Bölüm Konuları

- Giriş
- Alt Programların Temelleri
- Alt Programların Tasarım Problemleri
- Yerel Referans Platformları
- Parametre-Geçirme Metodları
- Altprogram Adı Olan Parametreler
- Altprogramları dolaylı Olarak Çağırma
- Aşırı Yüklenmiş Programlar
- Soysal Programlar
- Fonksiyonların Tasarım Modelleri
- Kullanıcı Tanımlı Aşırı Yüklenmiş Operatörler
- Kapatmalar
- Eşyordamlar

Giriş

- İki Temel Soyutlama Olanağı
  - İşlem Soyutlama
    - Erken Vurgulanmışlardır
    - Bu bölümde tartışıldı
  - Veri Soyutlama
    - 1980'lerde vurgulanmıştır
    - Bölüm 11'de uzunca tartışıldı

Altprogramların Temelleri

- Bir altprogramın tekbir giriş noktası vardır
- Çağrılan altprogramın yürütülmesi sırasında çağrılan askıya alınır
- Çağrılan altprogramın yürütülmesi sona erince kontrol daima çağırana döner

Temel Tanımlar

- Bir altprogramın
  - Python'da,fonksiyon tanımları yürütülebilir,diğer bütün dillerde fonksiyon tanımları yürütülemez
  - Ruby'de işlev tanımlarında veya sınıf tanımlarının dışındada görünebilir. Eğer dışarda ise, Object metodlarıdır. Nesne olmadan fonksiyon gibi çağırılabilirler
  - Lua'da bütün fonksiyonlar anonimdir
- Bir altprogram çağrısı altprogramın çalışması için belirtik bir istektir
  - Bir altprogram başlığı(subprogram header) tanımın (definition) ilk satırıdır; adı(name), altprogramın tipini, ve formal parametreleri içerir
- Bir altprogramın parametre profili o parametrenin sayısı,sırası ve türüdür.
  - Bir altprogramın(subprogram) protokolü(protocol) onun parametre profili artı, eğer bir fonksiyon ise, döndürdüğü tiptir (return type)

Temel Tanımlar(Devam)

- *C ve C++'* da fonksiyon bildirimlerine prototipler denir.
- Bir altprogram bildirimi(subprogram declaration) altprogramın protokolünü sağlar, ama gövdesini değil
- Bir formal parametre(formal parameter) altprogram başlığında (subprogram header) gösterilen ve altprogramda(subprogram) kullanılan bir kukla değişkendir (dummy variable)
- Bir etkin parametre (actual parameter) altprogram çağrı ifadesinde(subprogram call statement) kullanılan değer(value) veya adresi gösterir

Etkin/Biçimsel Parametrelerin Uygunluğu

- Konumsal(Pozisyonel)
  - Etkin parametrelerin biçimsel parametrelere bağlamı gereği: Birinci etkin parametre,birinci biçimsel parametreye bağlıdır,diğerleri de aynı şekilde
  - Güvenli ve Etkili
- Anahtar Sözcük
  - Etkin parametreye bağlı olan biçimsel parametrenin adı:etkin parametre ile belirtilir
  - *Avantaj*:*Advantage*: Parametreler herhangi bir sırada ortaya çıkabilir, böylece uygunluk hatalarından kaçınılmış olunur
  - *Dezavantaj*: Kullanıcı biçimsel parametrelerin ismini bilmek zorundandır

Biçimsel(Formal) Parametrelerin Varsayılan(Default) Değerleri

- Belirli dillerde (örn., C++, Python, Ruby, Ada, PHP), biçimsel parametreler varsayılan değerlere sahip olabilirler (Eğer hiçbir etkin parametre geçerli değil ise)
  - C++'da, varsayılan parametreler en son ortaya çıkar çünkü parametreler konumsal olarak birleştirilmiştir (anahtar sözcük parametre yok)

- Parametrelerin Değişken Sayıları
  - C# yöntemleri parametrelerin değişken sayıları olarak kabul edilebilir,uygun biçimsel parametrelerin **`params`** öncesinde bir dizi ile aynı olduğu sürece.
  - Ruby'de, etkin parametreler anahtar(hash) kalıp elemanı olarak gönderilir ve uygun etkin parametre yıldız işaretinden(asterisk) önce gelir.
  - Python'da, gerçek değerlerin listesi ve uygun etkin parametre asterisk(yıldız işareti) olarak isimlendirilir.
  - Lua'da, parametrelerin değişken sayıları biçimsel parametre olarak üç periottur; onlara **`for`** deyimiyle yada çoklu atama ile üç periottan ulaşılır

Ruby Blokları

- Ruby sıklıkla dizlerin eleman özellilerini kullanan yinelyici fonksiyonların sayılarını içerir
- Yineleyiciler tanımlı uygulamalar tarafından bloklarla sağlanır.
- Bloklar ilişik yöntem çağırımlarıdır; parametreleri olabilir (dikey çubuklarda); Yöntem `yield` deyimiyle uygulandığında onlarda uyguılanabilir

```
def fibonacci(last)
   first, second = 1, 1
   while first <= last
     yield first
     first, second = second, first + second
   end
end

puts "Fibonacci numbers less than 100 are:"
fibonacci(100) {|num| print num, " "}
puts
```

Prosedürler ve Fonksiyonlar

- Alt programların iki kategorisi vardır:
  - *Prosedürler* tanımlı parametreli hesaplamaların deyimlerinin toplamıdır
  - *Fonksiyonlar* yapısal olarak prosedürlere benzerler fakat yapı bakımından matematiksel fonksiyonlar üzerine modellenmişlerdir
    - Yan etki yaratmamaları beklenir
    - Pratikte,programın yan etkisi vardır

Altprogramların Tasarım Modelleri

- Yerel değişkenler(local variables) statik midir dinamik midir?
- Altprogram tanımları(subprogram definitions) diğer altprogram tanımlarında görünebilir mi?
- Hangi Parametre-Geçirme(Parameter-Passing) metodları sağlanmıştır?
- Parametre tipleri kontrol edilmiş midir?
- Geçen(passed) bir altprogramın referans platformu(referencing environment) nedir?
- Altprogramlar(Subprograms) aşırı-yüklenebilir mi(overloaded)?
- Altprogramların(Subprograms) soysal(generic) olmasına izin verilebilir mi?
- Eğer dil altprogramın yuvalanmasına izin verirse,kapatma bunu desteklermi?

Yerel Referans Platformları

- Eğer yerel değişkenler(local variables) yığın-dinamik(stack-dynamic) ise
  - Avantajlar
    - Özyineleme(recursion) desteği
    - Yereller(locals) için bellek bazı altprogramlar(subprograms) arasında paylaşılır
  - Dezavantajlar
    - Ayırma/Serbest Bırakma(Allocation/deallocation) süresi
    - Dolaylı Adresleme(Indirect addressing)
    - Altprogramlar(Subprograms) tarih duyarlı(history sensitive) olamaz
- Yerel Değişkenler statik olabilir
  - Avantajlar ve dezavantajlar statik-dinamik yerel değişkenlerin tam tersidie

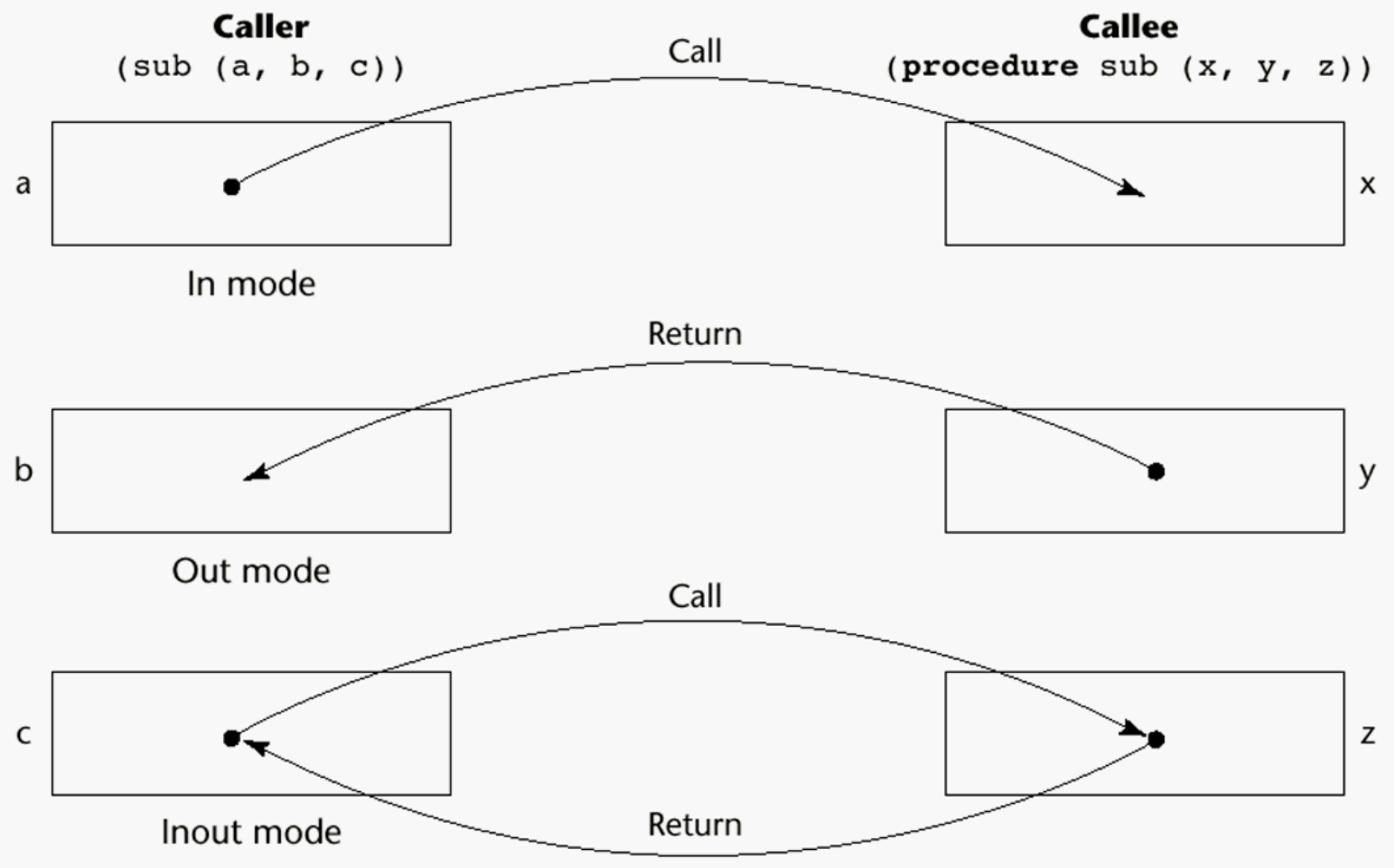
Yerel Referans Platformları: Örnekler

- Bir çok çağdaş dilde,yereller yığın dinamiktir
- C – her ikisi (değişkenler **`static`** olarak tanımlanan değişkenler)
- C++, Java, Python, C#'de sadece yığın dinamik yerel vardır
- Lua'da, bütün dolaylı bildirimli değişkenler küreseldir; yerel değişkenler `local` ile bildirilmiştir ve yığın dinamiktir

Parametre Geçişlerinde Semantik Modeller

- In mode
- Out mode
- Inout mode

Parametre Geçişlerinin Modelleri



Kavramsal Modellerin Transferi

- Değeri fiziksel taşıma
- Erişim yolu ile taşıma

Değeriyle Geçirme (In Mode)

- Etkin parametrenin değeri uygulanabilir biçimsel parametreyi sıfırlardı
  - Normalde kopyalama tarafından uygulanan
  - Erişim yoluyla uygulanabilir fakat tavsiye edilmez (zorlama yazı koruması kolay değil)
  - *Dezavantajları* (eğer fiziksel yolla taşınmışsa): Daha çok belleğe ihtiyaç duyar ve taşımaların maliyeti
  - *Dezvantajları*(eğer erişim yoluyla taşınmışsa): Çağrılmış altprogramda yazmaya korumalı olmalıdır ve maliyeti çoktur

Sonucuyla Geçirme (Out Mode)

- Yerelin değeri(local's value) çağırana(caller) geri gönderilir
- Genellikle fiziksel taşıma(Physical move) kullanılır
- Dezavantajlar:
  - Eğer değer(value) passed ise, zaman ve alan
  - Her iki durumda da, sıraya bağımlılık problem olabilir

- Potansiyel problemler:
  - `sub(p1, p1);` **hangisi biçimsel parametre ise tekrar kopyalanıp** `p1` **tarafından temsil edilecek**
  - `sub(list[sub], sub);` Programın balında yada sonunda adres listesini[sub] hesapla

Sonucuyla-Değeriyle Geçirme (inout Mode)

- Pass-by-value ve pass-by-result kombinasyonu
- Aynı zamanda pass-by-copy de denir
- Biçimsel parametrelerin yerel bellekleri vardır
- Dezavantajlar:
  - Pass-by-result'inkiler
  - Pass-by-value'inkiler

Referansıyla Geçirme (Inout Mode)

- Bir erişim yolunu(access path) geçme
- Aynı zamanda pass-by-sharing de denir
- Avantaj: geçiş(passing) işlemi verimlidir (kopyalama veya ikiye katlanmış bellek yoktur)
- Dezavantaj:
  - Erişimler daha yavaştır
  - Potansiyel istenmeyen yan etkiler (çarpışmalar)
  - İstenmeyen takma adlar (genişletilmiş erişim)

```
fun(total, total);  fun(list[i], list[j];  fun(list[i], i);
```

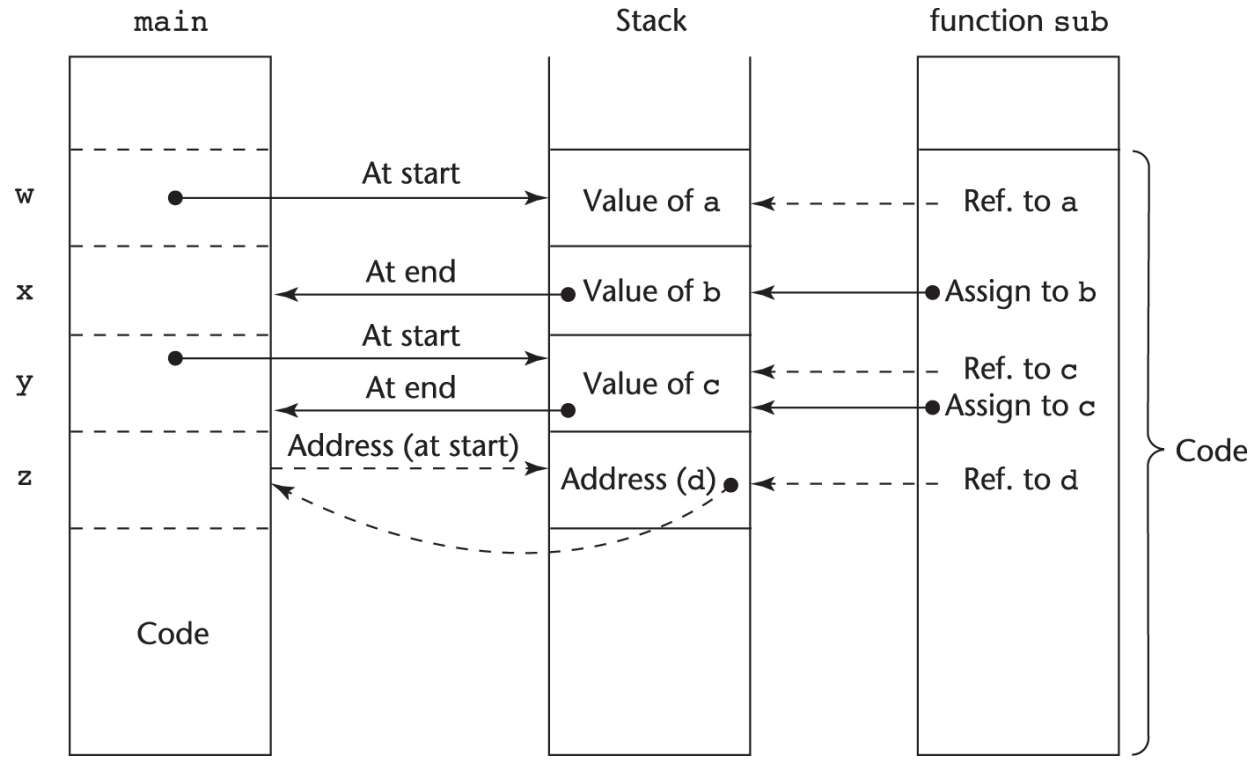
Adıyla Geçirme (Inout Mode)

- Metinsel ornatım(yerdeğiştirme) ile(textual substitution)
- Formaller çağrı(call) sırasında bir erişim metoduna (access method) bağlanır, fakat bir değere veya adrese asıl bağlama referans(reference) veya atama(assignment) sırasında olur
- İzinler geç bağlamlarda esnetilebilir
- Uygulamalar referansal platformun arayan parametresini geçirmesini gerektirir, böylece etkin parametre adresi hesaplanabilir

Parametre geçirmenin(Parameter Passing) implementasyonu

- Birçok dilde parametre iletişimi yığın süresinin yerini alıyor
- Referansıyla geçirmede uygulaması çok basit; sadece adres yığının yerini alır

Parametre-Geçirmenin (Parameter-Passing)Yığın(Stack) Implementasyonu



Fonksiyon başlığı: `void sub(int a, int b, int c, int d)`
Fonksiyon ana arama: `sub(w, x, y, z)`
(`w` 'değeri tarafından geç, `x` tarafından sonuç `y` taradından sonuç değeri, referans tarafından `z`)

Başlıca Dillerde Parametre Geçirme Metodları

- C
  - Değeriyle geçirme
  - Referansıyla geçirme işaretçileri parametre olarak kullanarak ulaşma
- C++
  - Refenransyıla geçirmede referansal tip denilen özel bir işaretçi türü
- Java
  - C++ gibidir, farkı sadece referanslar
- Ada
  - Üç semantik model de mümkündür: `in`, `out`, `in out`; `in` varsayılan moddur
  - "`out`"ise referans edilemez; "`in`" ise referans edilebilir fakat atanamaz; `in out` parametreleri referan edilebilir ve atanabilir

Başlıca Dillerde Parametre Geçirme Metodları(devamı)

- Fortran 95+
  - Parametreler in, out, yada inout modlarında etkinleştirilebilir
- C#
  - Varsayılan metod: değeriyle geçirme
    - Referansıyla geçirmede öncelikle biçimsel parametre ve etkin parametre `ref` ile belirtilmiştir
- PHP:C#'ye çok benzerdir, ikisinin de etkin yada biçimsel parametreleri ref ile belirtilmesi dışında
- Perl: bütün etkin parametreler dolaylı olarak önceden yerlerştirilmiş ve tanımlanmış dizi ismi `@_`
- Python ve Ruby atamayla geçirmeyi kullanır (bürün data değerleri nesnelerdir); etkin biçimsele atanmıştır

Tip Kontrol Parametreleri(Type checking parameters)

- Güvenilirlik için çok önemli olduğu düşünülmektedir
- FORTRAN 77 ve orjinal C: yok
- Pascal, FORTRAN 90+, Java, ve Ada: daima girektirir
- ANSI C and C++: kullanıcıya bırakılmıştır
  - Prototipler
- Nispeten yeni diller Perl, JavaScript, ve PHP tip kontrolünü gerektirmez
- In Python ve Ruby, değişkenlerin tipleri yoktur (nesnelerin vardır), yani parametre tip kontrolü mümkün değildir

Parametre olarak Çok boyutlu Diziler (Multidimensional Arrays)

- Eğer bir çok boyutlu dizi(multidimensional array) bir altprograma geçirilirse ve altprogram(subprogram) ayrı olarak derlenirse (compiled), derleyici(compiler) bellek eşleme fonksiyonunu(storage mapping function) oluşturmak(build) için o dizinin bildirilmiş boyutunu(declared size) bilmesi gerekir

Parametre olarak Çok boyutlu Diziler : C ve C++

- Programcı, etkin(actual) parametredeki ilk altsimge(subscript) dışında bütün belirtilmiş boyutları (declared sizes) dahil etmek zorundadır
- Bu esnek altprogramlar(Subprograms) yazmaya izin vermez
- Bu esnek altprogramlar(Subprograms) yazmaya izin vermez

Parametre olarak Çok boyutlu Diziler : Ada

- Ada – problem değildir
  - Kısıtlı Diziler – boyut dizi tipinin bir parçasıdır
  - Kısıtlı olamayan diziler– bildirilmiş boyut nesne bildiriminin(object declaration) parçasıdır

Parametre olarak Çok boyutlu Diziler : Fortran

- Biçimsel parametreler başlıktan sonra deklarasyonu(bildirim) olan dizilerdir
  - Tek için dizi boyutunda,indis alakasız(??)
  - Parametre olarak çoklu diziler için, biçimsel parametre bildirimleri geçirilmiş parametreler içerebilir,böylece bu değişkenler kullanılıp gönderim fonksiyonu olarak depolanabilir

Parametre olarak Çok boyutlu Diziler : Java and C#

- Ada ile benzerdir
- Diziler nesnelerdir; hepsi tek boyutlandırılmıştır fakat elemanlar diziler olabilir
- Her dizi adlandırılmış bir sabiti devralır (`length` Java'da, `Length` C#'de) dizi nesnesi yaratıldığında dizi uzunluğu ayarlanmış olur

Tasarımda Düşünülmesi Gerekenler

- İki önemli düşünce
  - Verimlilik
  - Tek yönlü veya çift yönlü
- Fakat bu iki düşüncede çelişkidir
  - İyi programlama => değişkenlere sınırlı erişim, yani mümkünse tek yönlü kullanım
  - Fakat referansıyla geçirme büyük yapıların geçirilmesinden daha verimlidir

Alt Programlar

- Altprogramların adının parametre oalrak geçmesi bazen uygundur
- Problemler:
  1. Parametrelerde tip kontrolü yapıldı mı?
  2. Parametre olarak gönderilmiş olan bir altprogramın doğru referans çevresi nedir?

Altprogram Parametre İsimleri

- *Yüzeysel Bağlama*: Platformu yasalaştıran (enacted) altprogram
  - Dinamik-kapsamlı dil için en doğaldır
- *Derin Bağlama*: Platformu tanımlayan alt program
  - Dinamik-kapsamlı dil için en doğaldır
- *Ad hoc bağlama*:Platformda çağrı deyiminin geçtiği altprogram

Altprogramları Dolaylı olarak Çağırma

- Genellikle çağrılacak birkaç olası alt program vardır ve doğru olanın çalışma süresi uygulanan kadar bilinmez (örn., olay giderme ve GUIs)
- C ve C++'da, bu tür çağrılar fonksiyon işaretçileriyle yapılır

Alt Programları Dolaylı Olarak Çağırma (devam)

- C#'de yöntem işaretçileri nesne olarak implementasyon(uygulandığında) edildiğinde buna *temsilciler(delegates)* denir
  - Bir temsilci(delagate) deklarasyonu:

```
public delegate int Change(int x);
```

    - Bu temsilci tipinde `Change` adında bir parametre, **`int` alan bir parametre ile örneklendirilebilir** ve **`int` değerine döner**

    Yöntem: `static int fun1(int x) { … }`

    Instantiate: `Change chgfun1 = new Change(fun1);`

    Denilebilir: `chgfun1(12);`

    - Bir temsilci bir adresten fazlasını depolayabilir,buna *çoğa gönderim temsilci (multicast delegate)*denir

Aşırı Yüklenmiş Altprogramlar

- Bir overloaded altprogram(subprogram) aynı referans çevresinde(referencing environment) bulunan diğer bir altprogramla aynı ada sahip olan altprogramdır
  - Aşırı yüklenmiş altprogramların bütün versiyonlarında benzersiz bir protokol vardır
- C++, Java, C#, ve Ada yerleşik aşırı yüklenmiş programlardır
- Ada'da, aşırı yüklü fonksiyonun dönüş türü belirsizliği gideren aramalar için kullanılabilir (böylece iki aşırı yüklenmiş fonksiyon aynı parameterlere sahip olabilir)
- Ada, Java, C++, ve C# altprogramların aynı ismi ile çoklu versiyonların yazımına izin verirler

Soysal Altprogramlar(Generic Subprograms)

- Bir soysal(generic) veya polimorfik altprogram(polymorphic subprogram) farklı etkinleştirmelerde farklı tipten parametreler alandır
- Aşırı-yüklenmiş altprogramlar(Overloaded Subprograms) ad hoc polymorphism sağlar
- *Subtype polymorphism* T türünde bir değişken tipi T herhangi bir nesne ya da T elde edilen herhangi bir tipe erişebilir (OOP dilleri) anlamına gelir
- Altprogamın parametrelerinin tiplerini tanımlayan bir tip ifadesinde kullanılan soysal bir parametre alan bir altprogram (subprogram) parametrik polimorfizm sağlar
  - Ucuz derleme-zaman yerine için dinamik bağlama

Soysal Altprogramlar(Generic Subprograms)(devam)

- C++
  - Fonksiyon bir çağrıda(call) adlandırıldığında veya adresi & operatörü ile çağrıldığında, C++ şablon(template) fonksiyonları örtük olarak(implicitly) başlatılır
  - Soysal altprogramlar **`template`** tarafından takip edilir soysal değişkenleri maddeler halinde tür isimleri ve sınıf isimlerine göre listelenir

```
template <class Type>
  Type max(Type first, Type second) {
  return first > second ? first : second;
  }
```

Soysal Altprogramlar(devam)

- Java 5.0
  - Java 5.0 ve C++'arasındaki soysal farklar:

  1. Java 5.0'de soysal parametreler sınıflandırılmış olmalıdır
  2. Java 5.0 'de soysal yöntemler sadece bir kere doğru olarak soysal yöntemler olarak örneklendirilmelidir
  3. Sınırlamalar sınıf aralıklarına göre belirtilmelidir ki soysal yöntemler soysal parametreler olarak geçebilsin
  4. Soysal parametrelerin joker türleri

Soysal Altprogramlar(devam)

- Java 5.0 (devam)

```
public static <T> T doIt(T[] list) { … }
```

  - parametreler soysal elemanların dizileridir (`T` türün ismidir)
  - Bir çağrım:

```
doIt<String>(myList);
```

Soysal parametrelerin sınırları olabiir:

```
public static <T extends Comparable> T
  doIt(T[] list) { … }
```

Soysal tür `Comparable` arabirimini uygulayan bir sınıf olmalıdır

Soysal Altprogramlar(devam)

- Java 5.0 (continued)
  - Joker türleri

    `Collection<?>` Derleme sınıfları için bir joker türüdür

```
void printCollection(Collection<?> c) {
    for (Object e: c) {
      System.out.println(e);
    }
 }
```

  - Herhangi bir derleme sınıfı için çalışabilir

Soysal Altprogramlar(devam)

- C# 2005
  - Java 5.0 ile benzer soysal yöntemleri destekler
  - Bir fark: etkin tür parametreler derleyici tanımlanmamış türüne ulaşabilirse,bir çağrı atlanabilir
  - Diğer – C# 2005 jokerleri(wildcard) desteklemez

Soysal Altprogramlar(devam)

- F#
  - Soysal tür dışında bir parametre türü veya bir işlemin dönüş türü saptanamıyorsa–*otomatik genelleştirme*
  - Böyle türler bir kesme işareti veya bir har ile belirtilir,örn., `'a`
  - Fonksiyonlar genel parametreleri tanımlayabilir

```
let printPair (x: 'a) (y: 'a) =
    printfn "%A %A" x y
```

    - `%A` her tür için format koddur
    - Bu parametreler kısıtlı tür değillerdir

Soysal Altprogramlar(devam)

- F# (devam)
  - Eğer parametrelerin fonksiyonları aritmetik operatörler ile kullanılıyorlarsa hatta parametreler soysal olarak belirtilse bile
  - Tür sonuç çıkarmaları(inferencing)ve tür zorlamalarının(coercions) eksikliği yüzünden, F# soysal fonksiyonları C++, Java 5.0+,ve C# 2005+'dan çok daha az kullanışlıdır

Fonksiyonların Dizayn Problemleri

- Yan etkilere izin verilir mi?
  - Çift-yönlü parametreler (Ada izin vermez)
- Hangi return tiplerine izin verilir?
  - Çoğu zorunlu diller return türlerini kısıtlar
  - C diziler ve fonksiyonlar türleri haricinde hiçbirine izin vermez
  - C++ ,C gibidir ancak sadece kullanıcı tanımlı türlere izin verir
  - Ada altprogramları herhangi bir türe dönebilir(return) (fakat Ada altprogramları tür değildir, yani dönemezler(return olamazlar))
  - Java ve C# yöntemleri herhangi bir türe return olabilir(dönüşebilir) (çünkü yöntemler tür değildir,döndürülemezler(return olamazlar))
  - Python ve Ruby yöntemleri birinci sınıf nesne gibi işleyebilir,yani diğer sınıfları kadar iyi dönebilirler(return olabilirler)
  - Lua fonksiyonların çoklu değerlere dönmesine(return olmasına) izin verir

Kullanıcı tanımlı Aşırı yüklü Operatörler

- Ada, C++, Python,ve Ruby'de kullancılar daha fazla operatör overload yapabilirler
- Python örneği:

```
def __add__ (self, second) :
  return Complex(self.real + second.real,
                 self.imag + second.imag)
```

Kullanım: hesaplama `x + y, x.__add__(y)`

Kapatmalar

- Kapatma(closure):bir altprogramın ve referansal platformun nerde tanımlandığıdır
  - Altprogram rastgele bir yerden çağrılabilir değilse referansal platform gereklidir
  - Statik-kapsamlı dil yuvalanmış altprogramlara izin vermez kapatmalara ihtiyacı yoktur
  - Kapatmalara sadece eğer altprogram yuvalanmış kapsamının içindeki değişkene erişebildiği zaman ihtiyaç duyulur ve herhangi bir yerden çağırılabilir
  - Kapatmaları desteklemek için, bir implemantasyonda(uygulamada)bazı değişkenlere sınırsız ölçüde değer verilebilir (çünkü bir altprogram normalde hayatta olmayan bir değişkene erişebilir)

Kapatmalar (devam)

- Bir JavaScript kapatması:

```
function makeAdder(x) {
  return function(y) {return x + y;}
}
...
var add10 = makeAdder(10);
var add5 = makeAdder(5);
document.write("add 10 to 20: " + add10(20) +
               "<br />");
document.write("add 5 to 20: " + add5(20) +
               "<br />");
```

  - Kapatılması `makeAdder` tarafından döndürülen adsız işlem

Kapatmalar (devam)

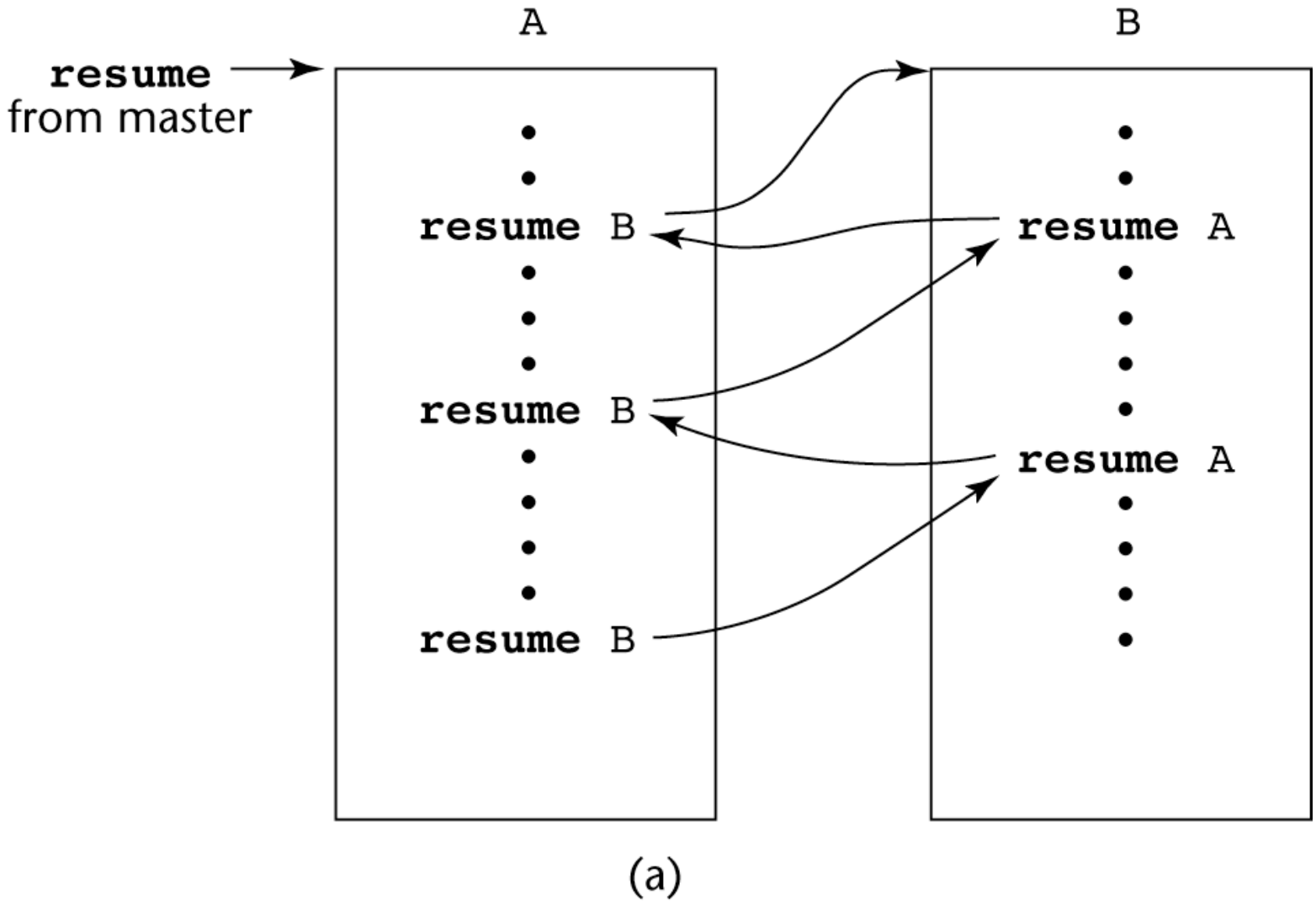
- C#
  - Yuvalanmış bir temsilci kullanarak C# 'de aynı kapatmayı yazabiliriz
  - `Func<int, int>` (return türü) specifies a delegate that takes an `int` as a parameter and returns and `int`

```
static Func<int, int> makeAdder(int x) {
   return delegate(int y) {return x + y;};
}
...
Func<int, int> Add10 = makeAdder(10);
Func<int, int> Add5 = makeAdder(5);
Console.WriteLine("Add 10 to 20: {0}", Add10(20));
Console.WriteLine("Add 5 to 20: {0}", Add5(20));
```

Eşyordamlar

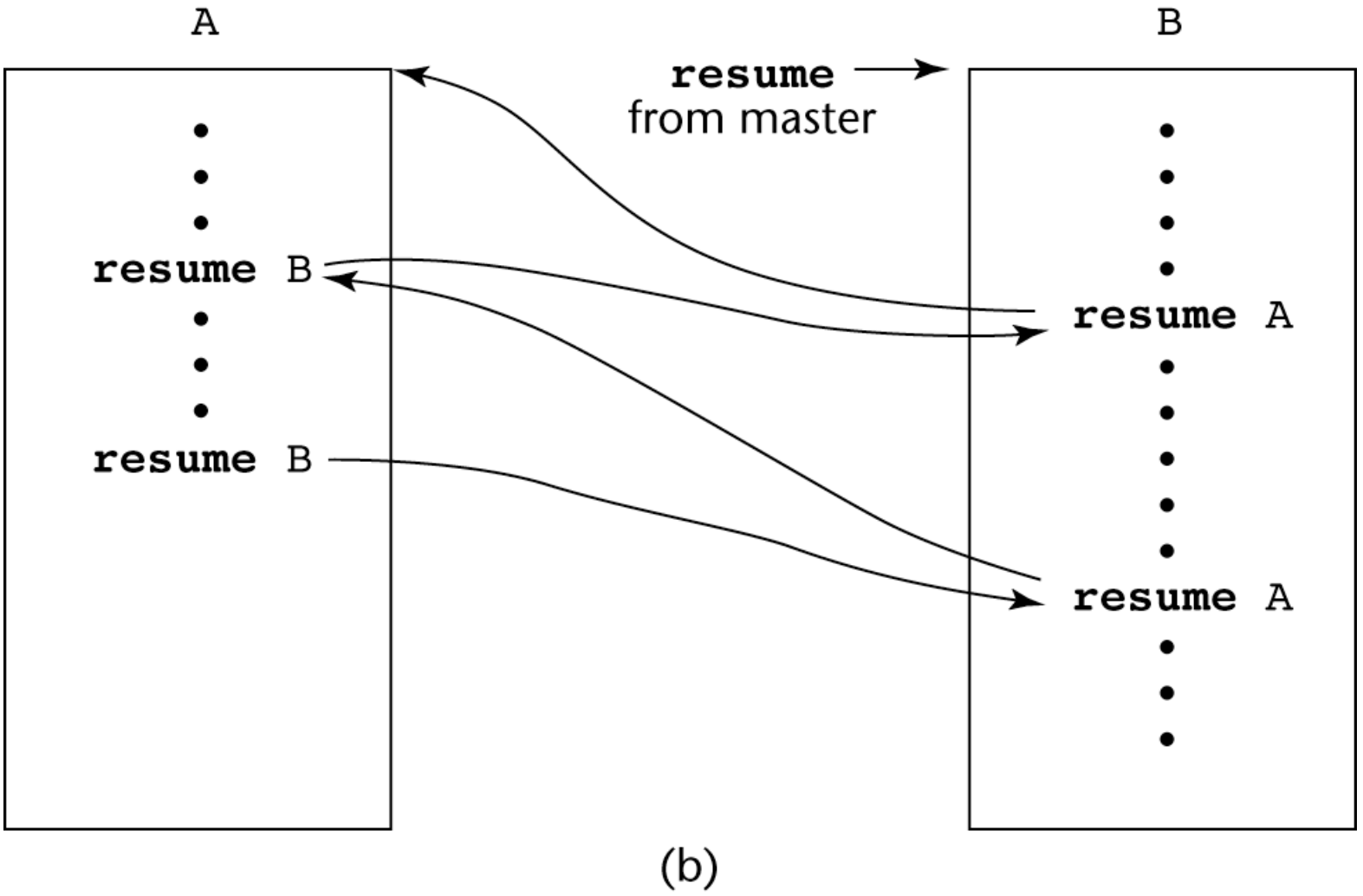
- Bir eşyordam(coroutine) birden çok girişi (entries) olan ve bunları kendi başına kontrol eden bir altprogramdır(subprogram) Lua'da
- Simetrik Kontrol(symmetric control) de denir
- Bir eşyordamın ilk sürdürmesi(resume) onun başlangıcınadır, fakat sonra gelen çağrılar(calls) eşyordamın en son çalıştırılan ifadesinden hemen sonraki noktadan girer
- Genellikle, eşyordamlar(coroutines) tekrar tekrar birbirini sürdürür(resume), belki sonsuza kadar
- Eşyordamlar(Coroutines) program birimlerinin(program units--(eşyordamlar)) eşzamanlıymış gibi yürütülmesini(quasi-concurrent execution) sağlar

Eşyordamlar



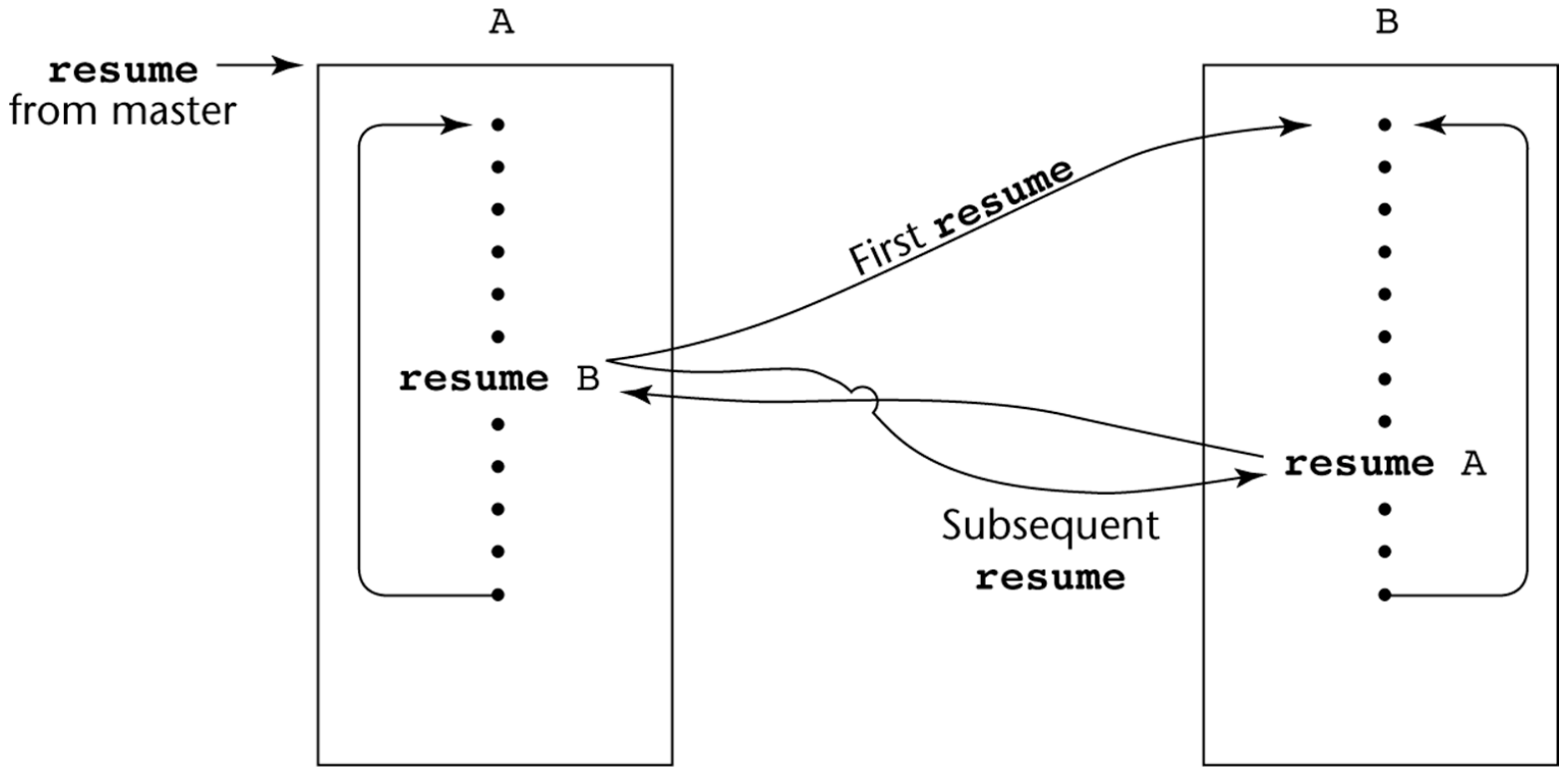
(a)

Eşyordamlar



(b)

Eşyordamlar



Özet

- Bir altprogram tanımlarken olay yine bir altprogram tarafından temsil edilir
- Altprogramlar fonksiyon veya prosedürde olabilir
- Altprogramlardaki yerel değişkenler statik yada dinamik olarak yığılabilirler
- Parametre geçmelerinde üç model vardır: in mode, out mode, ve inout mode
- Bazı dilller operatörlerin aşırı yüklenmesine izin verir
- Altprogramlar soysal olabilir
- Bir kapatma,altprogram ve onun referansal platformudur
- Bir eşyordam çoklu girişlerde bir altprogramdır