BMB214 Programlama Dilleri Prensipleri

Ders 10. Alt Programları Uygulama (Implementing Subprograms)

Erdinç Uzun
Tekirdağ Namık Kemal Üniversitesi
Bilgisayar Mühendisliği Bölümü

Konular

- Çağırma ve Return Genel Anlambilimi
- "Basit" Alt Programları Uygulama
- Alt Programları Stack-Dynamic Yerel Değişkenlerle Uygulama
- İç içe geçmiş alt programlar (Nested Subprograms)
- Bloklar (Blocks)
- Dinamik Kapsama (Dynamic Scoping) Uygulama

Çağırma (Call) ve Geri Dönüşlerin (Return) Genel Anlambilimi (Semantics)

- Bir dilin alt program çağırma (call) ve return işlemlerine birlikte alt program bağlantısı (subprogram linkage) denir
- Bir alt programa yapılan çağrıların genel anlambilim
  - Parametre geçirme yöntemleri
  - Yerel değişkenlerin stack-dynamic tahsisi
  - Çağıran programın yürütme durumunu kaydet
  - Kontrolün devri ve return düzenlenmesi
  - Alt program iç içe yerleştirme destekleniyorsa, yerel olmayan değişkenlere erişim düzenlenmelidir

Çağırma (Call) ve Geri Dönüşlerin (Return) Genel Anlambilimi (Semantics)

- Alt program return'lerin genel semantiği:
  - In ve inout modunda parametrelerin değerleri döndürülmelidir
  - Stack-dynamic yerellerin geri verilmesi
  - Yürütme durumunu (execution status) geri yükleyin
  - Kontrolü çağırana (caller) geri ver

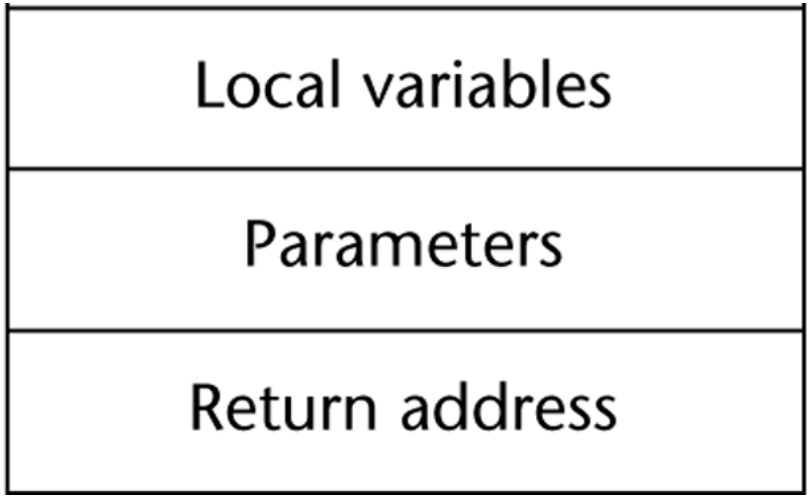
"Basit" Alt Programları Uygulama…

- Return Anlabilimi:
  - Pass-by-value-result veya out modu parametreleri kullanılıyorsa, bu parametrelerin mevcut değerlerini karşılık gelen gerçek parametrelere taşıyın.
  - Bir fonksiyon ise, fonksiyon değeri çağıranın (caller) alabileceği bir yere taşıyın
  - Çağıranın (Caller) yürütme durumunu geri yükleyin
  - Kontrolü çağırana geri aktar
- Gerekli depolama:
  - Durum bilgileri (Status information), parametreler, dönüş adresi (return address), fonksiyonlar için dönüş değeri, geçiciler (temporaries)

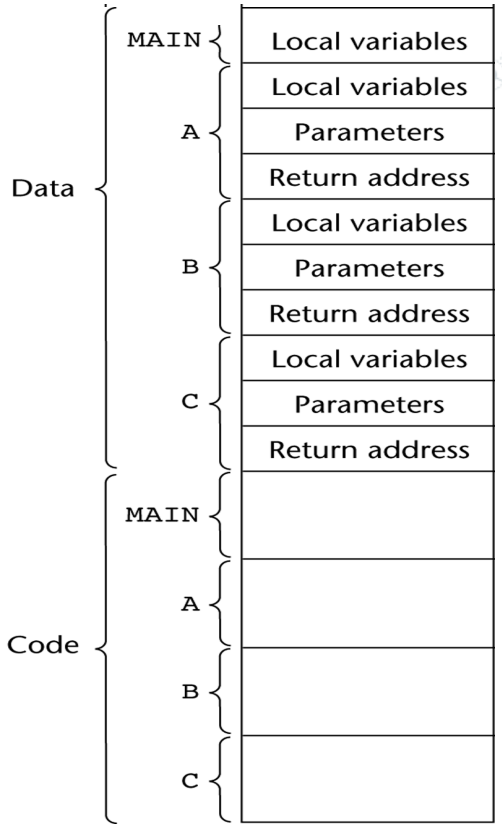
"Basit" Alt Programları Uygulama…

- İki ayrı bölüm: gerçek kod (actual code) ve kod olmayan (non-code) bölüm (yerel değişkenler ve değişebilen veriler)
- Yürütülen bir alt programın kod olmayan kısmının formatı veya düzeni, aktivasyon kaydı (activation record) olarak adlandırılır.
- Bir aktivasyon kaydı örneği (activation record instance), bir aktivasyon kaydının somut bir örneğidir (belirli bir alt program aktivasyonu için veri toplama)

"Basit" Alt Programları Uygulama…

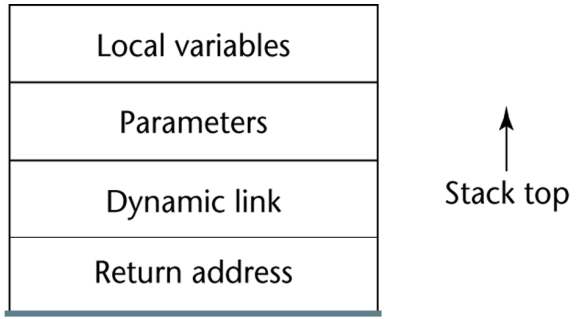


"Basit" Alt Programlara Sahip Bir Programın Kod ve Aktivasyon Kayıtları



Alt Programları Yığın Dinamik (Stack-Dynamic) Yerel Değişkenlerle Uygulama

- Daha karmaşık aktivasyon kaydı
  - Derleyici, yerel değişkenlerin örtük olarak ayrılmasına ve serbest bırakılmasına neden olmak için kod üretmelidir.
  - Yineleme desteklenmelidir (bir alt programın birden çok eşzamanlı (multiple simultaneous) aktivasyonu olasılığını ekler)

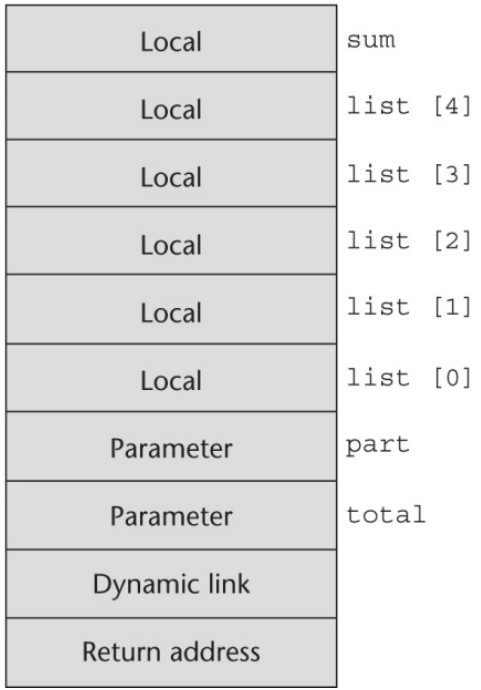


Alt Programları Stack-Dynamic Yerel Değişkenlerle Uygulama: Aktivasyon Kaydı (Activation Record)

- Aktivasyon kaydı biçimi (activation record format) statiktir, ancak boyutu dinamik olabilir
- Dinamik bağlantı (dynamic link), çağırana (caller) aktivasyon kaydının bir örneğinin üstüne işaret eder
- Bir alt program çağrıldığında bir aktivasyon kaydı örneği (instance) dinamik olarak oluşturulur
- Aktivasyon kaydı örnekleri, çalışma zamanı yığınında (stack) bulunur
- Ortam İşaretçisi (Environment Pointer - EP), çalışma zamanı sistemi (run-time system) tarafından korunmalıdır. Her zaman şu anda yürütülmekte olan program biriminin aktivasyon kaydı örneğinin tabanını gösterir.

C Fonksiyonu

```
void sub(float total, int part)
{
    int list[5];
   float sum;
    …
}
```



Revize Edilmiş Anlamsal Çağrı (Call) / Return İşlemleri

- Çağıran (Caller) İşlemleri:
  - Aktivasyon kaydı örneği oluşturun
  - Mevcut program biriminin yürütme durumunu kaydedin
  - Hesaplayın ve parametreleri iletin
  - Return adresini çağırana iletin
  - Kontrolü aranan kişiye aktarın
- Çağrılanın (Called) Prologue (Önsöz, başlangıç) eylemleri:
  - Eski EP'yi dinamik bağlantı olarak stack'e kaydedin ve yeni değeri oluşturun
  - Yerel değişkenleri tahsis edin

Revize Edilmiş Anlamsal Çağrı (Call) / Return İşlemleri…

- Çağrılanın Epilogue (son söz) eylemleri:
  - Pass-by-value-result veya out-mode parametreler varsa, bu parametrelerin mevcut değerleri ilgili gerçek parametrelere taşınır.
- Alt program bir fonksiyon ise, değeri çaıranın erişebileceği bir yere taşınır.
- Stack işaretçisini (pointer) mevcut EP-1'in değerine ayarlayarak geri yükleyin ve EP'yi eski dinamik bağlantıya (old dynamic link) ayarlayın
- Çağıranın yürütme durumunu geri yükleyin
- Kontrolü çağırana (caller) geri aktar

Özyineleme (Recursion) Olmayan bir Örnek

```
void fun1(float r) {
     int s, t;
     ...
     fun2(s);
     ...
}
void fun2(int x) {
     int y;
     ...
     fun3(y);
     ...
}
void fun3(int q) {
     ...
}
void main() {
     float p;
     ...
     fun1(p);
     ...
}
```

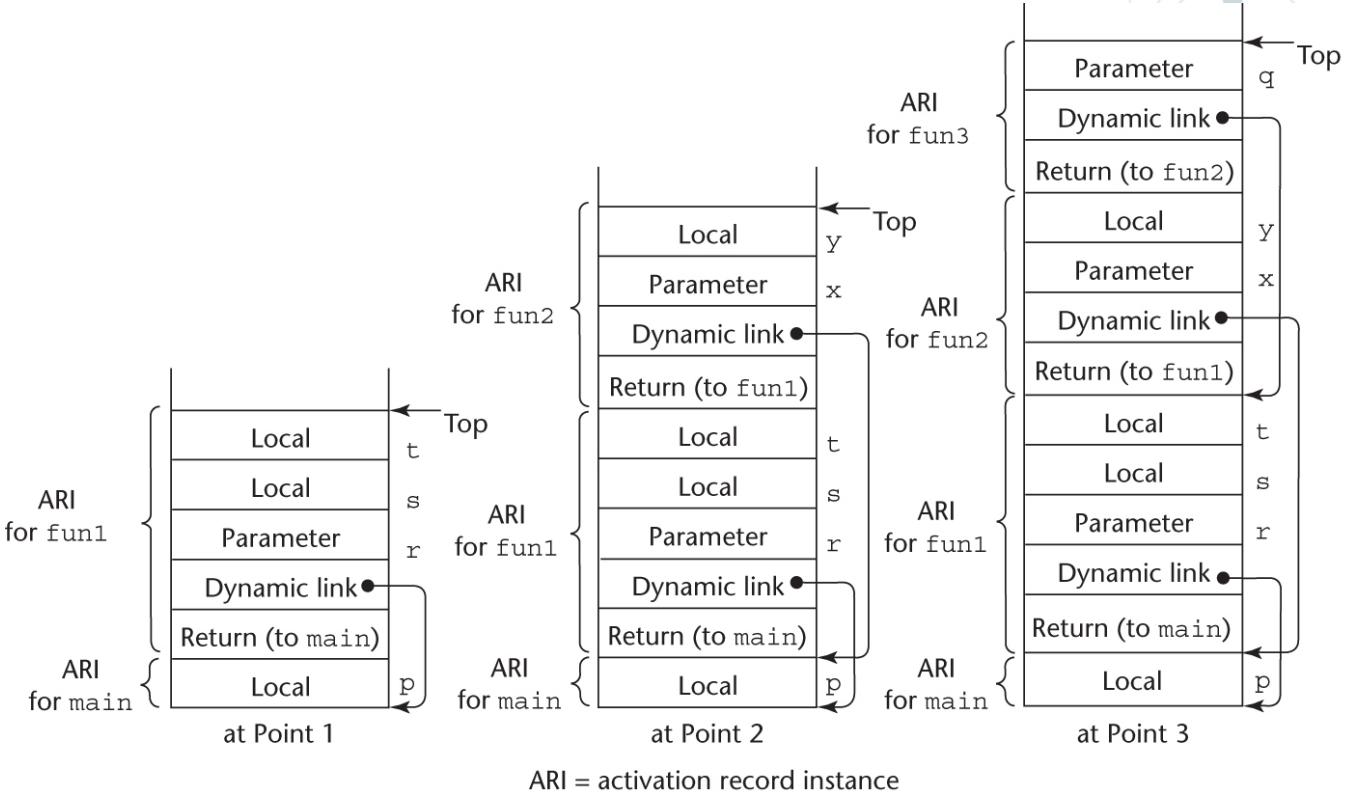
Main,fun1 çağırır

Fun1, fun2 çağırır

Fun2,fun3 çağırır

Özyineleme (Recursion) Olmayan bir Örnek

```
void fun1(float r) {
     int s, t;
     ...
     fun2(s);
     ...
}
void fun2(int x) {
     int y;
     ...
     fun3(y);
     ...
}
void fun3(int q) {
     ...
}
void main() {
     float p;
     ...
     fun1(p);
     ...
}
```



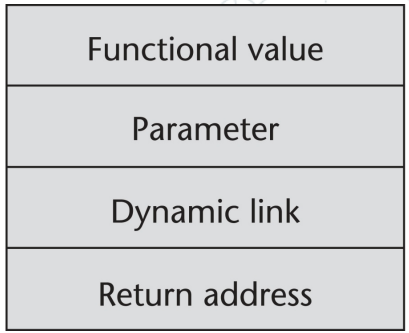
Dinamik Zincir ve Yerel Ofset (Dynamic Chain and Local Offset)

- Belirli bir zamanda stack’teki dinamik bağlantıların toplanması, dinamik zincir (dynamic chain) veya çağrı zinciri (call chain) olarak adlandırılır.
- Lokal değişkenlere, adresi EP'de bulunan aktivasyon kaydının başlangıcından itibaren ofsetiyle erişilebilir. Bu ofset, local_offset olarak adlandırılır
- Yerel bir değişkenin local_offset'i derleyici tarafından derleme zamanında (compile time) belirlenebilir

Özyineleme Örneği

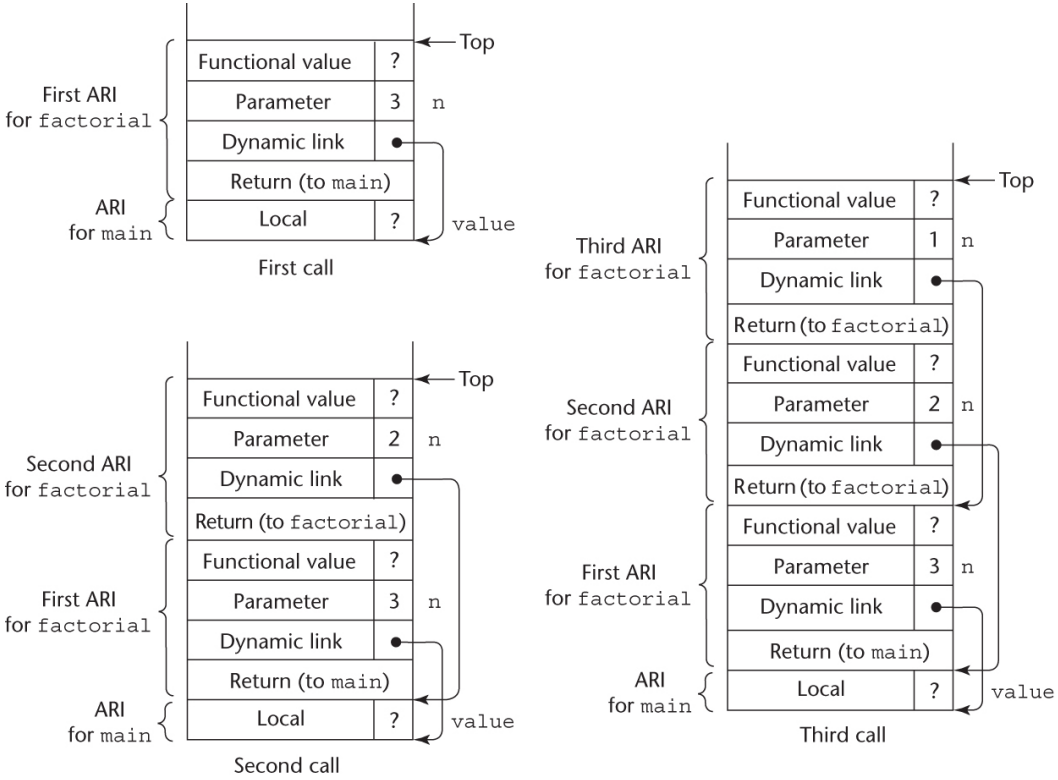
- Önceki örnekte kullanılan aktivasyon kaydı özyinelemeyi destekler

```
int factorial (int n) {
     <------------------------------1
   if (n <= 1) return 1;
   else return (n * factorial(n - 1));
     <------------------------------2
 }
 void main() {
   int value;
   value = factorial(3);
     <------------------------------3
 }
```



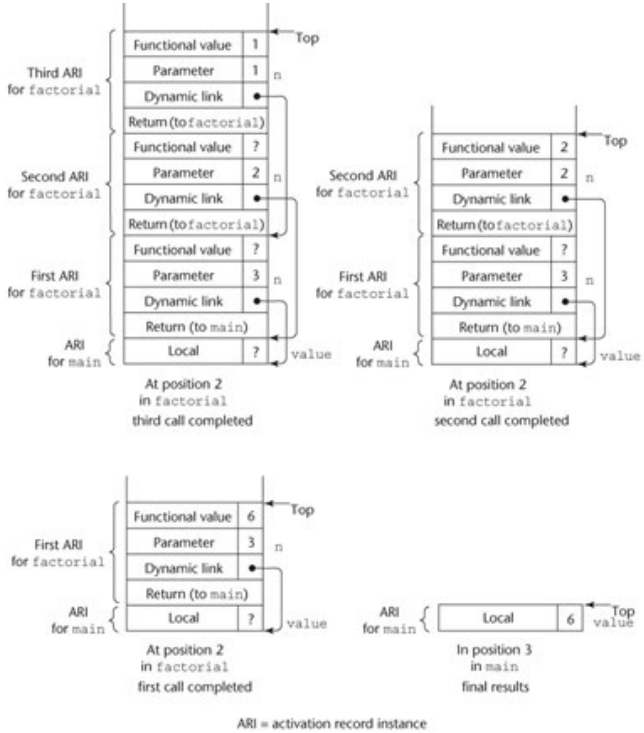
Factorial aktivasyon kaydı

Özyineleme Örneği…
Çağrılar (Call) için Stack'ler: `factorial`



ARI = activation record instance

Özyineleme Örneği…
Geri dönüş (Return) için Stack'ler : `factorial`



İç içe geçmiş alt programlar (Nested Subprograms)

- Örneğin, Fortran 95+, Ada, Python, JavaScript, Ruby ve Lua stack-dynamic yerel değişkenler kullanır ve alt programların iç içe geçmesine izin verir
- Yerel olarak erişilemeyen tüm değişkenler, stack'teki bazı aktivasyon kaydı örneğinde (activation record instance) bulunur.
- Yerel olmayan bir referansı bulma süreci:
  - Doğru aktivasyon kaydı örneğini bul
  - O aktivasyon kaydı örneğinde doğru ofseti belirle

Statik Kapsam Belirleme (Static Scoping)

- Statik zincir (static chain), belirli aktivasyon kaydı örneklerini birbirine bağlayan bir statik bağlantılar zinciridir.
- A alt programı için bir aktivasyon kaydı örneğindeki statik bağlantı (static link), A'nın statik ebeveyninin (parent) aktivasyon kaydı örneklerinden birine işaret eder.
- Bir aktivasyon kaydı örneğinden gelen statik zincir, onu tüm statik atalarına (ancestors) bağlar
- Static_depth, değeri o kapsamın iç içe geçme derinliği olan statik bir kapsamla ilişkili bir tamsayıdır.

Statik Kapsam Belirleme…

- Yerel olmayan bir referansın chain_offset veya nesting_depth'i, başvurunun static_depth'i ile bildirildiğinde kapsamınki arasındaki farktır.
- Bir değişkene referans, şu çift ile temsil edilebilir:

  (chain_offset, local_offset),

  burada local_offset, başvurulan değişkenin aktivasyon kaydındaki ofsettir

Javascript Örneği

```
function main(){
  var x;
  function bigsub() {
    var a, b, c;
    function sub1 {
      var a, d;
       a = b + c; <----------------------------1
      ...
    }  // end of sub1
    function sub2(x) {
      var b, e;
function sub3() {
       var c, e;
       ...
        sub1();
        ...
       e = b + a; <----------------------------2
      } // end of sub3 ...
```

main için call sırası

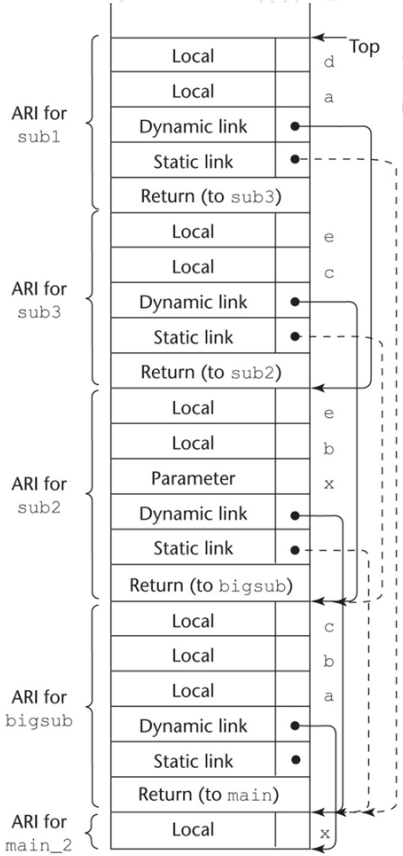
main, bigsub çağırır
bigsub, sub2 çağırır
sub2, sub3 çağırır
sub3, sub1 çağırır

```
sub3();
      ...
       a = d + e; <----------------------------
----3
    } // end of sub2
    ...
    sub2(7);
    ...
  } // end of bigsub
  ...
  bigsub();
  ...
} // end of main
```

Statik Zincir Bakımı (Static Chain Maintenance)

- Çağrıda(Call),
  - Aktivasyon kaydı örneği oluşturulmalıdır
  - Dinamik bağlantı (dynamic link) yalnızca eski stack üst (top) işaretçisidir
  - Statik bağlantı (static link), statik üst öğenin en son aktivasyon kaydı örneğini göstermelidir
  - İki metot:
    - Dinamik zinciri ara
    - Alt program çağrılarını ve tanımları değişken referanslar ve tanımlar gibi ele al

Position 1'de Stack İçeriği



ARI = activation record instance

Statik Zincir Bakımı (Static Chain Maintenance)

- Sorunlar:
  - Yerleştirme derinliği (nesting depth) büyükse yerel olmayan bir referans işlemi yavaştır
  - Zaman açısından kritik kod (Time-critical code) zordur:
    - Yerel olmayan referansların maliyetlerini belirlemek zordur
    - Kod değişiklikleri, yerleştirme derinliğini ve dolayısıyla maliyeti değiştirebilir

Bloklar (Blocks)

- Bloklar, değişkenler için kullanıcı tarafından belirlenen yerel kapsamlardır
- C Örneği

```
{
 int temp;
 temp = list[upper];
 list[upper] = list[lower];
 list[lower] = temp
}
```

- Yukarıdaki örnekte temp yaşam ömrü (lifetime), kontrol bloğa girdiğinde başlar.
- temp gibi yerel bir değişken kullanmanın bir avantajı, aynı ada sahip başka bir değişkenle etkileşime girememesidir.

Blok Uygulaması

- İki Metot:
  - Blokları her zaman aynı konumdan çağrılan parametresiz alt programlar olarak ele alın
    - Her bloğun bir aktivasyon kaydı vardır; blok her yürütüldüğünde bir örnek oluşturulur
  - Bir blok için gereken maksimum depolama alanı statik olarak belirlenebildiğinden, bu alan miktarı aktivasyon kaydındaki yerel değişkenlerden sonra tahsis edilebilir.

Dinamik Kapsam Belirleme (Dynamic Scoping) Uygulama

- Local değişkenlere ve local olmayan referanslara dynamic-scoped dillerde iki yol vardır. (Emacs Lisp, SNOBOL and APL)
  - Derin Erişim (Deep Access): yerel olmayan referanslar, dinamik zincirdeki aktivasyon kaydı örnekleri aranarak bulunur
    - Zincirin uzunluğu statik olarak belirlenemez
    - Her aktivasyon kaydı örneğinin değişken adları olmalıdır
  - Gölge Erişim (Shallow Access): Yerel değişkenleri merkezi bir yere koyun
    - Her değişken adı için bir stack
    - Her değişken adı için bir giriş içeren merkezi tablo

Dinamik Kapsam Belirlemeyi Uygulamak İçin Derin Erişimin (Deep Access) Kullanılması

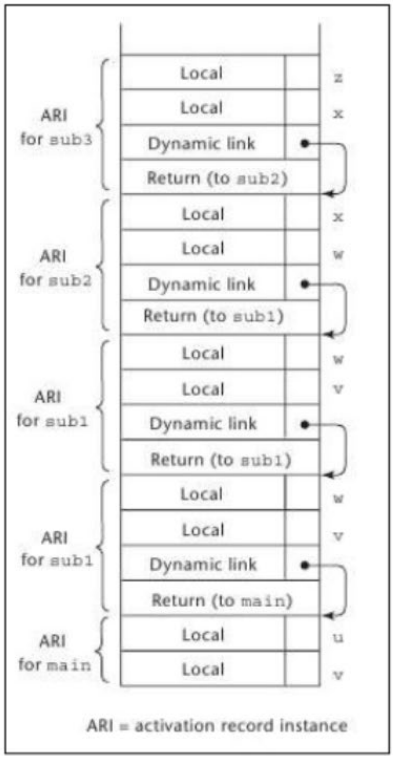
```
void sub3() {
  int x, z;
  x = u + v;
  ...
}

void sub2() {
  int w, x;
  ...
}

void sub1() {
  int v, w;
  ...
}

void main() {
  int v, u;
  ...
}
```

main calls sub1
sub1 calls sub1
sub1 calls sub2
sub2 calls sub3



Dinamik Kapsam Belirlemeyi Uygulamak İçin Gölge Erişimin (Shallow Access) Kullanılması

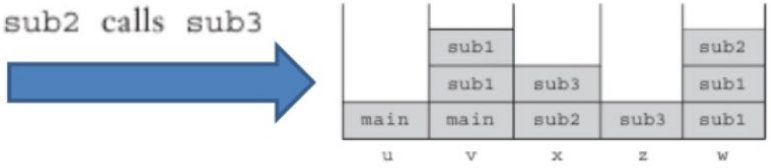
```
void sub3() {
  int x, z;
  x = u + v;
  ...
}

void sub2() {
  int w, x;
  ...
}

void sub1() {
  int v, w;
  ...
}

void main() {
  int v, u;
  ...
}
```

main calls sub1
sub1 calls sub1
sub1 calls sub2
sub2 calls sub3



Deep Access vs. Shallow Access

- Deep Access, alt programlarda bağlantıya hızlandırma sağlar, diğer taraftan local olmayan değişkenlere referans maliyetlidir.
- Shallow Access, local olmayan değişkenlere referansı hızlandırma sağlar, fakat alt programlara bağlantı çok maliyetlidir.