10. Alt Programların Gerçekleştirimi

Alt program çağrı ve geri dönüş işlemlerinin tamamına alt program bağlama adı verilir. Tipik bir programlama dilinde bir alt program çağrısına bağlı olab birçok olay vardır. Çağrı süreci, kullanılan parametre geçiş yönteminin gerçekleştirimini içermelidir. Yerel değişkenler statik değilse çağrı ile alt programdaki yerel değişkenler için bellekte yer ayrılır ve o yerler değişkenlerle bağlanır. Çağrıyı yapan program biriminin çalışma durumunu tutmak zorundadır. Çalışma durumu, çağrıyı yapan program biriminin çalışmasını sürdürebilmesi için gerekli olan herşeyi içerir: Register değerleri, CPU durum bitleri, EP(environment pointer) gibi. EP, alt programın çalışması esnasında parametrelere ve yerel değişkenlere erişmek için kullanılır.

Alt program çağrı süreci kontrolün alt programa geçişini düzenlemeyi ve çalışması bitince kontrolün uygun yere dönüşünü garantilemeyi de kapsar. Eğer bir dil iç içe alt programları destekliyorsa, alt programın yerel olmayan ancak kendisine görünür olan değişkenlere erişim mekanizması da çağrı süreci tarafından oluşturulur.

Alt programın dönüşü için gereken işlemler çağrıdakinden daha az karmaşıktır. Eğer out veya inout modda parametreler varsa dönüş süreci ilk olarak formal parametrelerin yerel değerlerini gerçek parametrelere aktarır. Sonra yerel değişkenler için ayrılan bellek bölgesini geri bırakır ve çağıran program biriminin çalışma durumunu geri yükler. En son, kontrol çağıran program birimine geri dönmelidir.

10.1. Basit Alt Programların Gerçekleştirimi

Basit kelimesi ile iç içe olmayan ve tüm yerel değişkenleri statik olan alt programlardan bahsediyoruz. Fortran' ın ilk sürümlerinde bu tip alt programlar vardı.

Basit alt programa çağrı için şu işlemler gerekir:

1) O an çalışan program biriminin çalışma durumunu saklama
2) Parametreleri hesaplama ve geçirme
3) Çağrılana dönüş adresi geçirme
4) Kontrolü çağrılana aktarma

Basit bir alt programda dönüş yapılırken,

1) Eğer out mode parametreler varsa bu parametrelerin değerleri ilgili gerçek parametrelere taşınır.
2) Eğer alt program bir fonksiyonsa, fonksiyonun değeri çağıranın erişebileceği bir yere taşınır.
3) Çağıranın çalışma durumu geri yüklenir.
4) Kontrol çağırana transfer edilir.

Çağrı ve dönüş işlemleri aşağıdakiler için depolama gerektirir.

- Çağıranın durum bilgisi
- Parametreler
- Dönüş adresi
- Fonksiyonlar için dönüş değeri
- Alt program tarafından kullanılan geçiciler

Bunların yanında yerel değişkenler ve alt program kodu, bir alt programın çalışması ve çağrıldığı yere dönmesi için gerekli tüm bilgiyi oluşturur.

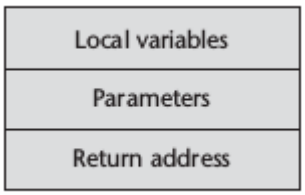
Yukarıda saydığımız işlerin çağıran ve çağrılan arasında nasıl paylaşılacağı konusu önemlidir. Basit alt programlar için bu sorunun cevabı çoğu iş için açıktır. Çağrı işleminin son üç adımı çağıran tarafından gerçekleştirilir. Çağıranın çalışma durumunu her iki taraf da saklayabilir. Dönüş adımlarını düşünürsek ilk 3. ve 4. adımlar çağrılan tarafından gerçekleştirilir. Yine çalışanın çalışma durumunun geri dönüşü iki taraftan birinin gerçekleştirebileceği bir adımdır.

Genellikle çağrılanın bağlama işlemleri iki farklı zamanda geçekleşir: Çalışmanın başında (prologue) ve sonunda (epilogue). Basit alt programlarda çalışanın bağlama işlemleri çalışmasının sonunda gerçekleştirilir.

Basit bir alt program iki parçadan oluşur: alt programın gerçek kodu (sabittir) ve yerel değişkenler ve alt programın çalışmasıyla değişebilecek diğer şeyler.

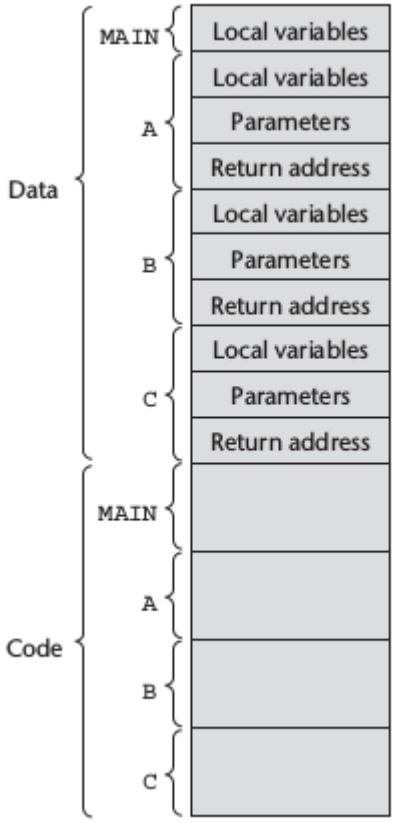
Kod olmayan kısım için format veya model Aktivasyon Kaydı (activation record) olarak adlandırılır. Çünkü bu kısımdaki veri sadece alt program aktifken (çalışması sürüyorken) alt programla ilgilidir. Aktivasyon kaydının belirli bir örneğine aktivasyon kayıt örneği (activation record instance) denir.

Basit alt programları olan dillerde özyineleme desteği olmadığından, bir alt programın belli bir zamanda sadece bir tane aktif sürümü olabilir. Dolayısıyla basit bir alt programın aktivasyon kaydı (Şekil 1) tek bir örnekten oluşur.



Şekil 1. Basit alt programlar için aktivasyon kaydı

Basit bir alt programın aktivasyon kaydı sabit büyüklükte olduğunda statik olarak yer tahsisi yapılabilir. Aslında kod kısmına eklenebilir.



Şekil 2. Basit alt programları olan bir programın kodu ve aktivasyon kaydı

Şekil 2' deki program bir main ve A, B, C olmak üzere üç tane alt programdan oluşuyor. Örnekte aktivasyon kaydı koda eklenmiştir.

10.2. Stack-dynamic yerel değişkenlere sahip alt programların gerçekleştirimi

Stack-dinamik yerel değişkenleri olan alt programlarda recursion desteği bulunur.

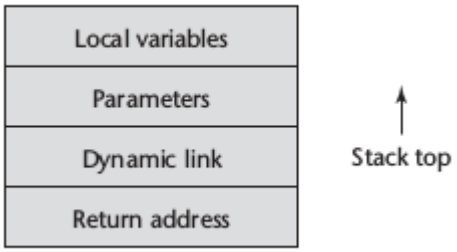
10.2.1. Daha karmaşık aktivasyon kaydı

Aşağıdaki sebepleren dolayı, stack-dinamik yerel değişken kullanımı sözkonusu olduğunda basit alt programlara göre bağlama daha karmaşıktır.

- Yerel değişkenlerin kapalı olarak bellek tahsisi ve geri bırakma için derleyici kod üretmek zorundadır.
- Özyineleme varsa alt programın eş zamanlı çoklu aktivasyonu sözkonusu olur ve belli bir zamanda birçok aktivasyon kaydı örneği bulunabilir.

Bir alt programın aktivasyon kaydının formatı çoğu dilde derleme zamanında bilinir. Çoğu durumda aktivasyon kayıtları için boyut da belirlidir çünkü tüm yerel veri sabit büyüklüktedir.

Stack- dinamik yerel değişkenleri olan dillerde aktivasyon kayıt örnekleri dinamik olarak yaratılmalıdır.

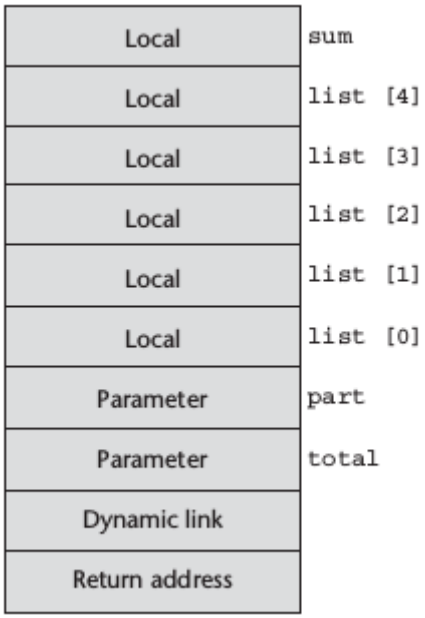


Şekil 3. Stack-dinamik değişlenleri olan bir alt program için tipik aktivasyon kaydı.

Şekil 3' te stack-dinamik değişkenleri olan bir alt program için aktivasyon kaydı görülmektedir. Dönüş adresi (return address), code segmentte çağrıyı yapan program biriminin çağrıdan bir sonraki komutunu işaret eden bir işaretçiden oluşur. Dinamik link çağıranın aktivasyon kayıt örneğinin tabanını işaret eden bir işaretçidir. Statik kapsamlı dillerde bu link çalışma zamanı hatalarının oluşması halinde geri dönmeyi sağlayan bilgiyi içerir. Dinamik kapsamlı dillerde dinamik link yerel olmayan değişkenlere erişim için kullanılır. Aktivasyon kaydında gerçek parametreler ya değerler ya da adreslerdir ve çağıran tarafından sağlanırlar.

Aşağıdaki C kodunun aktivasyon kayıt örneği Şekil 4' te verilmiştir.

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



Şekil 4 – sub fonksiyonu için aktivasyon kayıt örneği

Bir alt programı aktifleştirmek, o programa ait aktivasyon kaydının dinamik bir örneğini yaratmayı gerektirir. En son çağrılan alt programın ilk önce sonuçlanması gerektiğinden bu aktivasyon kayıt örneklerini bir stack içinde tutmak mantıklıdır. Bu stack, çalışma zamanı sisteminin bir parçası olduğundan adı run-time-stack' tir. Her alt program aktivasyonu için stack' ta aktivasyon kaydının yeni bir örneği yaratılır.

Alt programın işletilmesinin kontrolü için bir şey daha gereklidir: EP (environment pointer). Başlangıçta EP ana programın aktivasyon kayıt örneğinin tabanını veya ilk adresini gösterir. Run-time sistemi, EP' nin o anda aktif olan programın aktivasyon kayıt örneğinin tabanını göstermesini sağlamak zorundadır. Yeni bir alt program çağrılınca mevcut EP yeni oluşturulan aktivasyon kayıt örneğine dinamik link olarak yazılır. Sonra yeni aktivasyon kayıt örneğinin tabanını göstermek üzere ayarlanır. Alt programdan dönüşte stack' in en üstü (top) EP – 1' e eşitlenir ve EP de çalışmasını tamamlamış olan alt programın aktivasyon kayıt örneğindeki dinamik link' e eşitlenir. Bu durumda, stack' in top değerini değiştirmek en üstteki kaydı silmek anlamına gelmektedir.

Bağlama (linkage) sürecinde yeni işlemler gördük. Bu işlemlerle daha önceki bilgilerimizi güncelleyelim:

Çağıran işlemleri:
1) Aktivasyon kayıt örneği oluşturma
2) O an çalışan program biriminin çalışma durumunu saklama
3) Parametreleri hesaplama ve geçirme
4) Çağrılana dönüş adresi geçirme
5) Kontrolü çağrılana aktarma

Çağrılanın prologue işlemleri:
1) Eski EP değerini stack'te dinamik link olarak saklama ve yeni değer yaratma
2) Yerel değişkenlere yer tahsis etme

Çağrılanın epilogue işlemleri:
1) Eğer out mode parametreler varsa bu parametrelerin değerleri ilgili gerçek parametrelere taşıma
2) Eğer çağrılan bir fonksiyonsa, fonksiyonun değerini çağıranın erişebileceği bir yere taşıma
3) Stack pointer' ın değerini EP-1 yapma ve EP' yi eski dinamik link'e eşitleme
3) Çağıranın çalışma durumu geri yükleme
4) Kontrol çağırana transfer etme

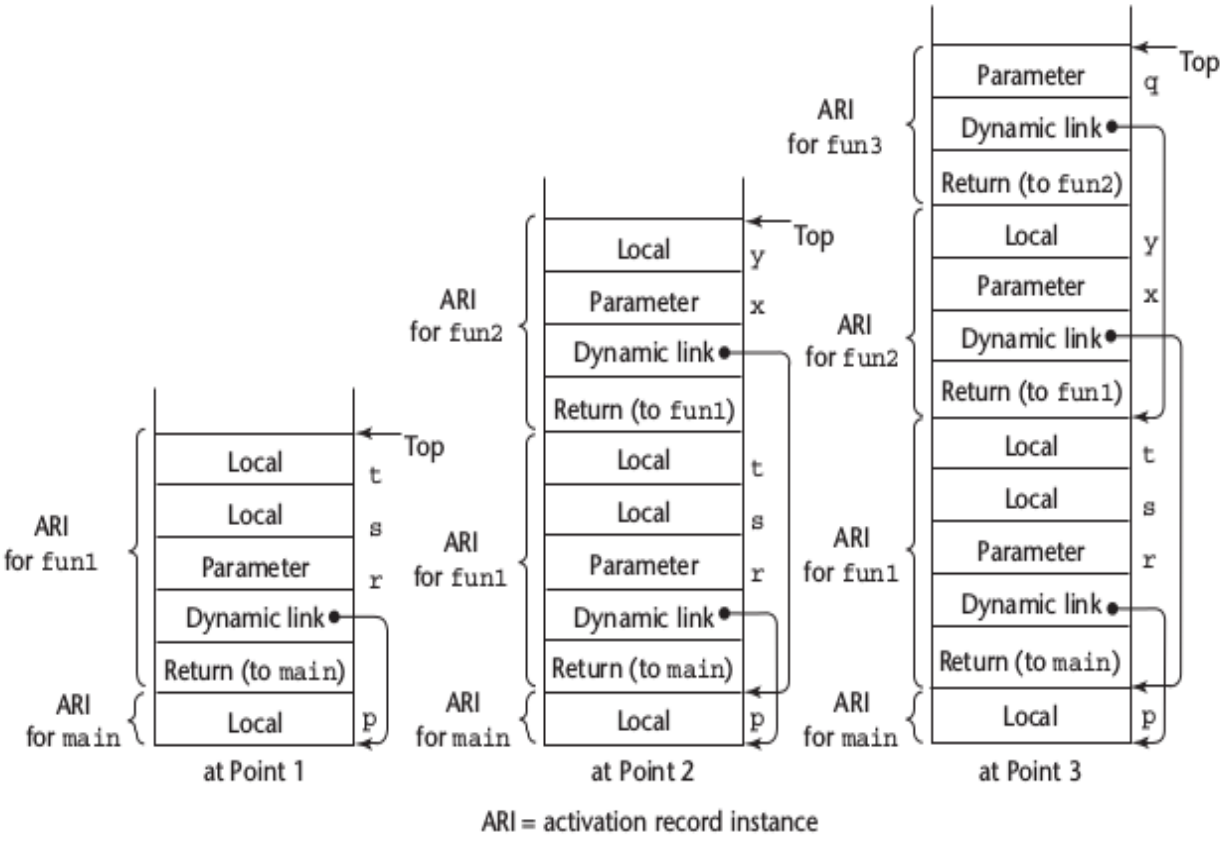
10.2.2. Özyinelemesiz Örnek

```
void fun1(float r) {
  int s, t;
  ...      <-------------- 1
  fun2(s);
  ...
}

void fun2(int x) {
  int y;
  ...      <-------------- 2
  fun3(y);
  ...
}

void fun3(int q) {
  ...      <-------------- 3
}

void main() {
  float p;
  ...
  fun1(p);
  ...
}
```

Yukarıdaki kodda fonksiyon çağırma sırası şöyledir:

main fun1' i
fun1 fun2' yi
fun2 fun3' ü çağırmaktadır.

Kodda 1, 2 ve 3 ile etiketlenmiş noktalarda stack durumu Şekil 5' te gösterilmiştir. 1. noktada sadece main ve fun1' e ait aktivasyon kayıt örnekleri stack' ta bulunmaktadır. Fun1, fun2' yi çağırdığında fun2' ye ait aktivasyon kayıt örneği de stack' a eklenir. Benzer şey fun2, fun3' ü çağırdığında da yapılır. Aktivasyon kayıt örneklerindeki dinamik link, o alt programın çağrıldığı yere ait olan aktivasyon kaydının tabanını gösterir. Fun3' ün çalışması bitince, ona ait olan aktivasyon kayıt örneği stack' tan silinir ve EP stack' ın en üst noktasını güncellemek için kullanılır.



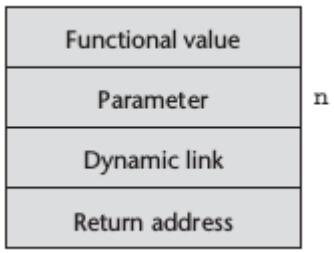
Şekil 5 – Programdaki üç noktada stack içeriği

Belirli bir zamanda stack' ta mevcut olan dinamik linklerin tümüne dinamik zincir veya çağrı zinciri denir. Bize çalışmanın bulunduğu noktaya nasıl geldiğini gösteren dinamik bir tarihçe bilgisi verir.

10.2.3. Özyinelemeli Örnek

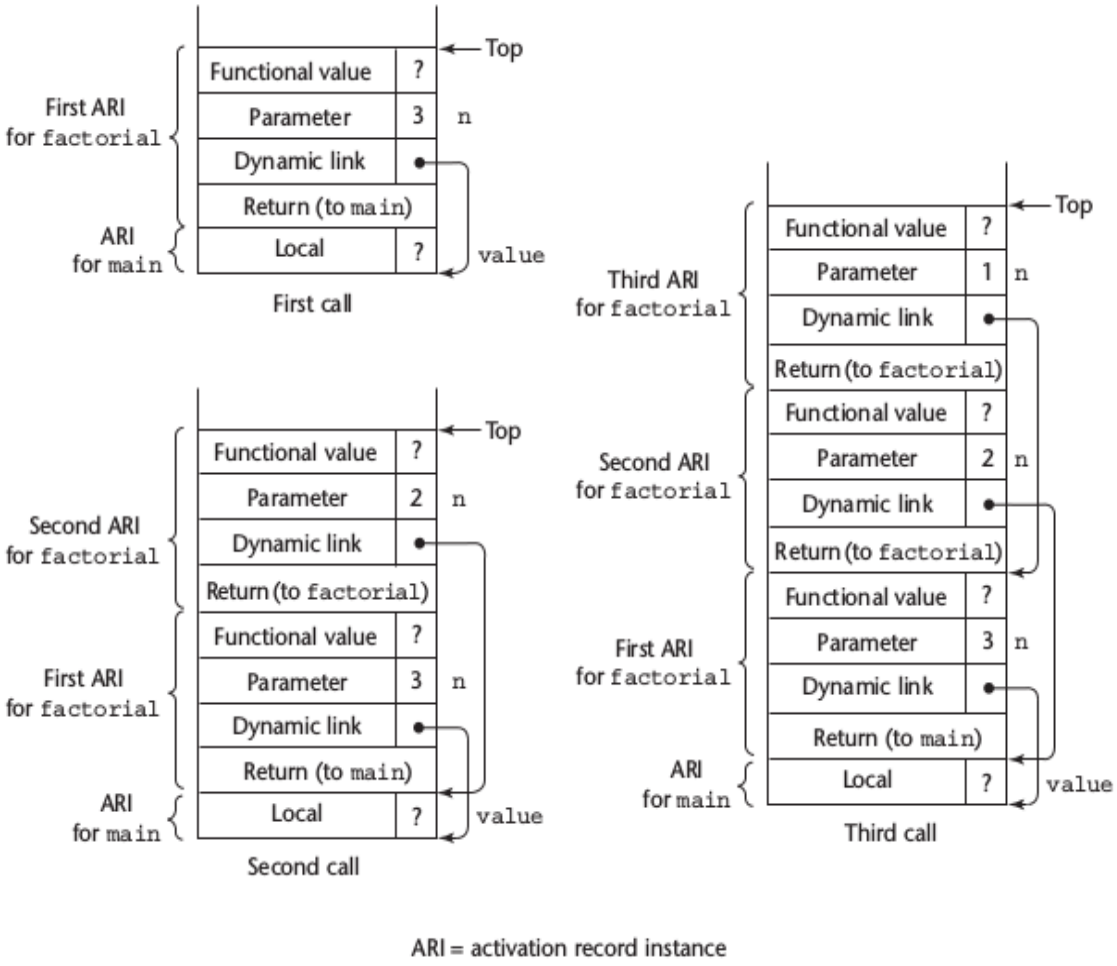
```
int factorial(int n) {
    <-------------- 1
  if (n <= 1)
    return 1;
  else return (n * factorial(n - 1));
    <-------------- 2
 }
void main() {
  int value;
  value = factorial(3);
    <-------------- 3
 }
```

Özyinelemeli fonksiyonlar için kullanılan aktivasyon kayıt örneği Şekil 6' daki gibidir.



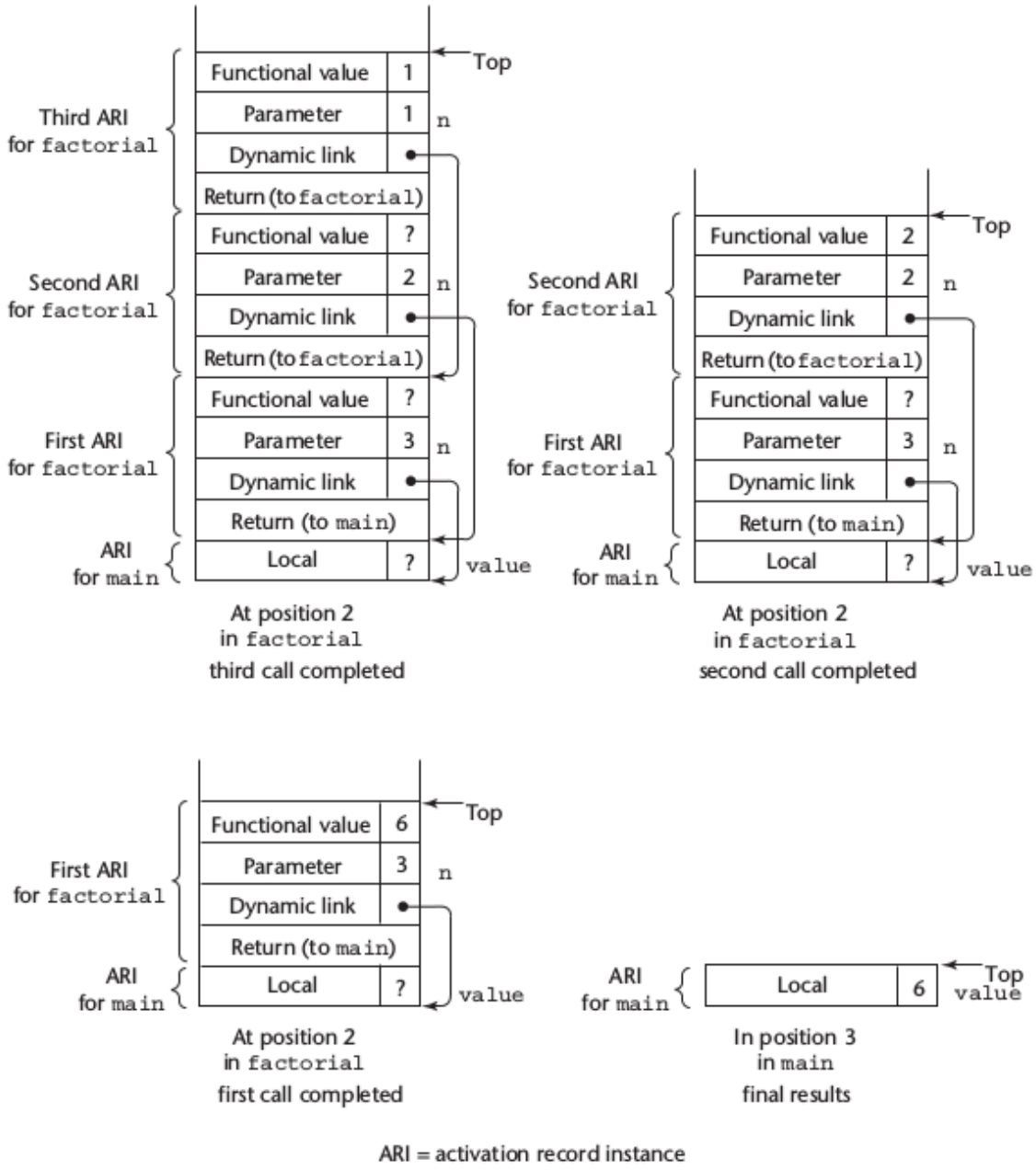
Şekil 6 – Özyinelemeli fonksiyonlar için aktivasyon kayıt örneği

Yukarıda verilen kodda özyinelemeli fonksiyon üç kere çağrılmaktadır. Bu üç çağrıya ait stack görünümü Şekil 7' de verilmiştir.



Şekil 7 – Koddaki 1 ile işaretlenmiş nokta için özyinelemeli fonksiyonun 3 çağrısına ait stack görünümü

Özyinelemeli fonksiyonlardan herhangi bir dönüş yapılmadığı sürece function value kısımları ? Ile işaretlenmiştir. Kodda 2 ve 3 ile etiketlenmiş noktalar için stack görünümü Şekil 8' deki gibidir.



Şekil 8 – main ve factorial' in çalışması süresince stack görünümü

10.3 İç içe alt programlar

C-tabanlı olmayan statik kapsamlı bazı programlama dilleri stack-dinamik değişken kullanır ve iç içe alt program tanımlamaya izin verirler. Bunlardan bazıları Fortran 95+, Ada, Python, JavaScript, Ruby ve fonksiyonel dillerdir.

10.3.1 Temel Bilgi

Statik kapsamlı iç içe altprogramlarda, yerel olmayan bir değişkene referans, iki adımlı bir erişim süreci içerir. Yerel olmayan ve statik olmayan tüm değişkenler mevcut aktivasyon kayıt örnekleri içindedir yani stack' tadır. İstenen değişkenin yerleştirildiği aktivasyon kayıt örneği ilk adımda bulunur. İkinci adımda da değişkenin yerel offsetini kullanarak aktivasyon kayıt örneği içindeki yerine erişilir.

Bu iki adımdan zor olanı doğru aktivasyon kayıt örneğini bulmaktır. Öncelikle şunu hatırlayalım: bir alt program için sadece statik atalarının (alt programın içinde tanımlandığı alt programlar) kapsamlarında tanımlanmış değişkenler görülebilir ve erişilebilirdir. Aynı zamanda alt programın tüm statik kapsamlı atalarına ait aktivasyon kayıt örnekleri de stack' ta bulunmaktadır. Bir program ancak tüm ataları aktifse çağrılabilir.

10.3.2 Statik Zincirler

Bu yaklaşımda statik link denen yeni bir işaretçi aktivasyon kaydına eklenir. Statik link (static scope pointer), statik ebeveynin aktivasyon kayıt örneğinin en tabanını gösterir. Yerel olmayan değişkenlere erişim için kullanılır. Stack' taki statik linklerin tamamına da statik zincir denir.

Bir alt programın statik derinliği, en dıştaki kapsamdan ne kadar uzakta olduğunu gösteren bir tamsayıdır. Hehangi bir alt program tarafından kapsanmayan alt programların statik derinliği sıfırdır.

```
# Global scope
. . .
def f1():
    def f2():
        def f3():
            . . .
        # end of f3
        . . .
    # end of f2
    . . .
# end of f1
```

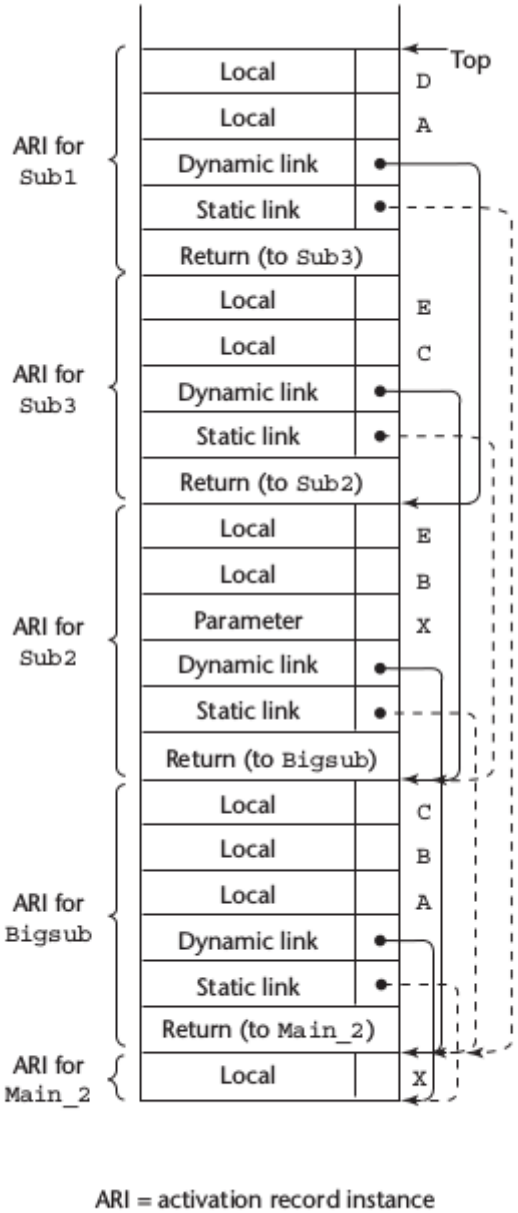
Yukarıdaki örnekte, global scope, f1, f2 ve f3' ün statik derinlikleri sırasıyla 0, 1, 2, 3' tür. Eğer f3 içinden f1' de tanımlanmış bir değişkene referans verilirse bu referans için **zincir offset**' i 2 olur. Zincir offseti, referansın verildiği değişkenin kaç seviye yukarıda tanımlandığıdır. Eğer f3 içinden f2' de tanımlı bir değişkene referans verilirse de o referansın zincir offseti 1 olur.

Yerel olmayan değişkenlere erişim için aşağıdaki Ada koduna bakalım.

```
procedure Main_2 is
  X : Integer;
  procedure Bigsub is
    A, B, C : Integer;
    procedure Sub1 is
      A, D : Integer;
      begin  -- of Sub1
      A := B + C;   <---------- 1
      ...
    end;  -- of Sub1
    procedure Sub2(X : Integer) is
      B, E : Integer;
      procedure Sub3 is
        C, E : Integer;
        begin  -- of Sub3
        ...
        Sub1;
        ...
        E := B + A; <---------- 2
      end;  -- of Sub3
      begin  -- of Sub2
      ...
      Sub3;
      ...
      A := D + E;   <---------- 3
    end;  -- of Sub2
    begin  -- of Bigsub
    ...
    Sub2(7);
    ...
  end;  -- of Bigsub
  begin  -- of Main_2
  ...
  Bigsub;
  ...
end;  -- of Main_2
```

Program çağrıları sırasıyla şöyledir:
main Bigsub' ı
Bigsub Sub2' yi
Sub2 Sub3' ü
Sub3 Sub1' i çağırmaktadır.

1 ile etiketlenmiş nokta için stack görünümü Şekil 9'daki gibidir. Statik link alt programın ebeveynine (içinde tanımlandığı alt program) ait aktivasyon kaydının tabanını gösterir.



Şekil 9 – Kodda 1 nolu noktadaki stack görünümü

Kodda 2 ile etiketlenmiş noktada E değişkenine stack içinde erişim için (0,4) gösterimi kullanılır. Burada 0, zincir offseti ve 4 'te yerel offsettir. E değişkenine 2. noktada sub 3 içinden referans verilmiştir. E değişkeninin tanımlandığı yer ise sub3' ün kendisidir. Dolayısıyla zincir offseti 0' dır. Yerel offset ise E değişkeninin tanımlı olduğu sub3' e ait aktivasyon kaydı içindeki yeridir. Saymaya kaydın tabanından başlanarak (taban 0 değerinde) E'nin bulunduğu yerin offseti 4 olarak bulunur.

A değişkeninin tanımlandığı yerler
1. noktada (0,3) – yerel değişken
2. noktada (2,3) – 2 seviye uzaktaki değişken
3. noktada (1,3) – 1 seviye uzaktaki değişken