1) *Ders07 - Pointer and Reference Types, Type Checking: (Sayfa 22 ve sonrası özet çıkarmanın gereksiz olmasından dolayı çoğunlukla eklenmedi !!!)*

- ***Tip Kontrolü,*** bir operatörün işlenenlerinin uyumlu tipte olmasını sağlama faaliyetidir.
- ***Uyumlu bir tür,*** operatör için yasal olan ya da dil kurallarının, derleyici tarafından oluşturulan kodla, örtük olarak dönüştürülmesine izin verilen bir yasal türe uygundur. Bu otomatik dönüşüme zorlama denir. Örneğin, Java'da bir float değişkeni ile int değişkeni toplandığında, int değişkeni float haline getirilir ve sonuç bu şekilde hesaplanır.
- ***Tip hatası,*** bir operatörün uygun olmayan bir işlenene uygulanmasıdır.
- ***Yazım tipi statik ise***, tüm tip denetimi derleyici tarafından statik olarak yapılabilir. Dinamik tip bağlaması, çalışma zamanında dinamik tip kontrolü gerektirir, örn. Javascript ve PHP
- Daha önceki düzeltme genellikle daha az maliyetli olduğu için, çalışma zamanı yerine derleme zamanında hataları saptamak daha iyidir.
- **Tür Uyumluluğu;**
  - ✓ İki değişkenin uyumlu tipte olmasının en önemli sonucu, her ikisinin de diğerine atanan değeri olabilir. Tip uyumluluğunu kontrol etmek için iki yöntem:

- Name Type Compatibility:
  - Two variables have compatible types only if they are in either the same declaration or in declarations that use the same type name.
- Advantage: Easy to implement
- Disadvantage: Highly restrictive
  - Subranges of integer types are not equivalent with integer types
  - Formal parameters must be the same type as their corresponding actual parameters

- Under a strict interpretation, a variable whose type is a subrange of the integers would not be compatible with an integer type variable
- Example:

```
type indexType = 1..10; {subrange type}
var count: integer;
index: indexType;
```

- The variables `count` and `index` are not name type compatible, and cannot be assigned to each other

Structure Type Compatibility

- Two variables have compatible types if their types have identical structure.
- Disadvantage: Difficult to implement
- Advantage: more flexible
- The variables `count` and `index` in the previous example, are structure type compatible.
- Structure type compatibility also disallows differentiating between types with the same structure

```
type celsius = float;
fahrenheit = float;
```

- They are compatible according to structure type compatibility but they may be mixed

  - ✓ Çoğu Programlama Dili bu yöntemlerin bir kombinasyonunu kullanır.
  - ✓ C, yapısal eşdeğerliği (structural equivalence), C++ ise isim eşdeğerliğini (name equivalence) kullanır.
  - ✓ Ada, isim uyumluluğunu kullanır, aynı zamanda iki tip yapı sağlar; -Subtypes, -Derived Types.

- **Derived Types:** Uyumsuz olduğu önceden tanımlanmış bazı türlere dayalı yeni bir tür. Ana türün tüm özelliklerini miras alırlar.

```
type celsius is new float
type fahrenheit is new float
```

- Yapıları aynı olmasına rağmen, iki tip birbiriyle uyumsuzdur. Ayrıca diğer kayar nokta (float) türleriyle de uyumsuzdurlar.
- **Subtypes:** Muhtemelen mevcut bir türün kısıtlı sürümünü gösterir. Bir alt tip ebeveyn türüyle uyumludur.
- `subtype small_type is Integer range 0..99;`
- Small_type değişkenleri tam sayı değişkenleriyle uyumludur.

❖ **C'de Tür Uyumluluğu;**

- ✓ C, ***structures*** ve ***unions*** dışındaki her tür için yapı türü uyumluluğunu kullanır.
- ✓ Her structures ve unions, başka hiçbir türle uyumlu olmayan yeni bir tür yaratır.
- ✓ Typedef'in herhangi bir yeni tip getirmediğini, ancak yeni bir isim tanımladığını unutmayın.
- ✓ C ++ isim eşdeğerliğini kullanır.

❖ **İç İçe Seçiciler (Nesting Selectors);**

- Java example

```
if (sum == 0)
   if (count == 0)
       result = 0;
else result = 1;
```

- Which **if** gets the **else**?
- Java's static semantics rule: **else** matches with the nearest **if**

- To force an alternative semantics, compound statements may be used:

```
if (sum == 0) {
  if (count == 0)
      result = 0;
}
else result = 1;
```

- The above solution is used in C, C++, and C#

- Python

```
if sum == 0 :
   if count == 0 :
     result = 0
   else :
     result = 1
```

2) ***Ders08 - Subprograms:***

❖ **Alt Programların Temelleri (Fundamentals of Subprograms);**

- ✓ Her alt programın tek bir giriş noktası vardır.
- ✓ Arama programı askıya alınmış durumda, çağrılan alt programın yürütülmesi halinde.
- ✓ Bu nedenle, belirli bir zaman diliminde yalnızca bir alt program yürütülmekte.

❖ **Temel Tanımlar;**

- ✓ Bir ***alt program tanımı,*** alt programı soyutlamanın eylemlerini ve arayüzünü açıklar.
- ✓ Python'da fonksiyon tanımları çalıştırılabilir; diğer tüm dillerde çalıştırılamaz.

```
if ….
  def fun1(…);
else
  def fun2(…);
```

- ✓ ***Alt program başlığı (subprogram header),*** adın, alt programın türünün ve biçimsel parametrelerin de dahil olduğu tanımın ilk bölümüdür.
- ✓ FORTRAN örneği >>> SUBROUTINE adı (parametreler), C örneği >>> void adder (parametreler)
- ✓ Bir ***alt program çağrısı (subprogram call),*** alt programın yürütülmesi için açık bir istektir.
- ✓ Bir ***alt programın parametre profili (aka signature),*** parametrelerinin sayısı, sırası ve türleridir.
- ✓ ***Protokol*** bir alt programın parametre profilidir ve eğer bir fonksiyonsa, dönüş tipidir.
- ✓ C ve C ++ 'daki işlev bildirimlerine genellikle ***prototipler (prototypes)*** denir.
- ✓ Bir ***alt program bildirimi (subprogram declaration),*** alt programın gövdesini değil, protokolünü sağlar.
- ✓ ***Resmi bir parametre (formal parameter),*** alt program başlığında listelenen ve alt programda kullanılan kukla (dummy) bir değişkendir.
- ✓ ***Gerçek bir parametre (actual parameter),*** alt program çağrısı deyiminde kullanılan bir değeri veya adresi temsil eder.

❖ **Parametreler;**

Example in Ada,

```
SUMER (LENGTH => 10,
       LIST => ARR,
       SUM => ARR_SUM);
```

Formal parameters: LENGTH, LIST, SUM.

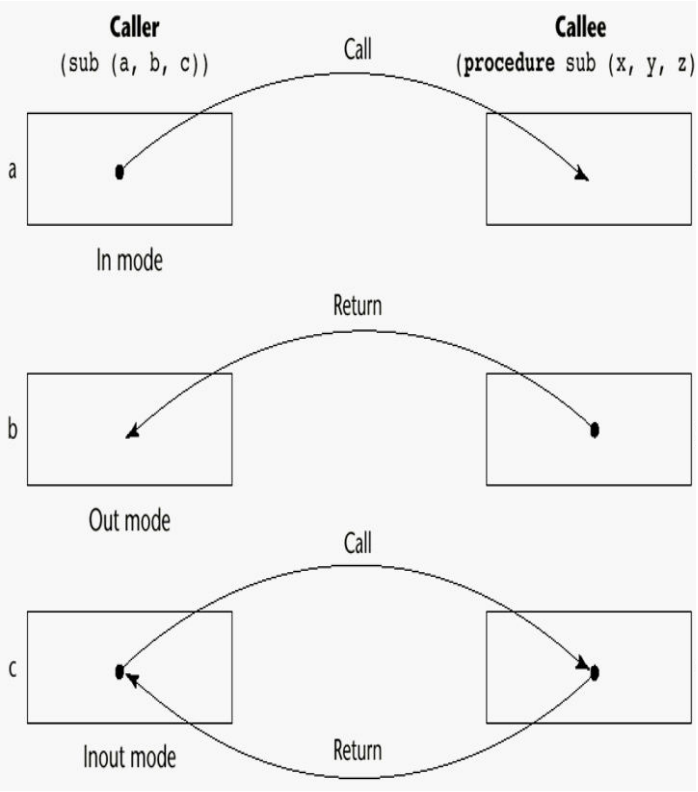
Actual parameters: 10, ARR, ARR_SUM.

- Ada, and FORTRAN 90 allow positional parameters and keyword parameters to be used together.

```
SUMER (10, SUM => ARR_SUM, LIST => ARR);
```

- Once a keyword appears in the call, all remaining parameters must be keyword parameters.

❖ **Parameter Passing; (https://courses.cs.washington.edu/courses/cse505/99au/imperative/parameters.html)**



- ***Pass-by-Value (In Mode);*** Bu yöntemde in-mode kullanır. Resmi parametrede yapılan değişiklikler arayana geri iletilmez. Çağrılan yöntem içindeki biçimsel parametre değişkeninde yapılan değişiklikler yalnızca ayrı depolama konumunu etkiler ve çağrı ortamındaki gerçek parametreye yansıtılmaz. Bu yöntem aynı zamanda değere göre çağrı olarak da adlandırılır.
- ***Pass-by-Reference (In/Out Mode);*** Bu teknik, iç/dış mod anlambilimini kullanır. Resmi parametrede yapılan değişiklikler, arayan (caller) parametreye geri iletilir. Formal parametredeki herhangi bir değişiklik, formal ortamdaki gerçek verilere bir referans aldığı için, çağrı (caller) ortamındaki gerçek parametreye yansıtılır. Bu yöntem hem zaman hem de mekanda etkilidir.
- ***Pass-by-Result (Out Mode);*** Bu yöntem out-mode anlamını kullanır. Kontrol arayan kişiye geri aktarılmadan hemen önce, formal parametrenin değeri gerçek parametreye geri iletilir. Bu yönteme bazen sonuçtan çağrı çağrılır. Genel olarak, sonuç tekniğine göre geçiş kopya ile uygulanır.
- ***Pass-by-Value Result (In/Out Mode);*** Bu yöntem, iç / dış mod anlamını kullanır. Bu, Pass-by-Value ve bir Pass-by-Result yöntemlerinin bir kombinasyonudur. Kontrol arayan kişiye geri aktarılmadan hemen önce, formal parametrenin değeri gerçek parametreye geri iletilir. Bu yönteme bazen değer-sonuç (value-result) çağrısı denir.

- ***Pass-by-Name (In/Out Mode);*** Bu teknik Algol gibi bir programlama dilinde kullanılır. Bu teknikte, bir değişkenin sembolik "adı" geçirilir ve bu hem erişilmesine hem de güncellenmesine izin verir. Bağımsız değişken ifadesi, formal parametre her iletildiğinde yeniden değerlendirilir. Prosedür, argüman ifadesinde kullanılan değişkenlerin değerlerini değiştirebilir ve böylece ifadenin değerini değiştirebilir. Yandaki örnekte de görülebileceği üzere "call by name" çağrısında fonksiyonun parametresine "n+10"un değeri değil, direkt olarak ismi geçiriliyor. Yani "k" ifade bir nevi "n" ile değiştirilmiş oluyor. Bundan sonraki işlemlerde "k"yi "n" olarak düşünebiliriz.

```
begin
integer n;
procedure p(k: integer);
   begin
   print(k);
   n := n+1;
   print(k);
   end;
n := 0;
p(n+10);
end;

Output:

call by value:   10 10
call by name:    10 11
```

❖ **Referencing Environment;**

- ✓ **Soru:** Geçirilen alt programı yürütmek için Başvuru Ortamı (Referencing Environment) nedir? (Lokal olmayan değişkenler için)
  - ***Sığ ciltleme (Shallow Binding):*** Geçirilen alt programı etkileyen call ifadesinin ortamı. Dinamik-kapsamlı diller için en doğal durum.
  - ***Derin ciltleme (Deep Binding):*** Geçirilen alt programın tanımı ortamı. Statik kapsamlı diller için en doğal.
  - ***Geçici ciltleme (Ad Hoc Binding):*** Alt programı geçen call deyiminin ortamı.

```
Example:
function sub1() {
  var x;           2:declared in        Passed subprogram S2
  function sub2() {                     Output:
    window.status = x;                  is called by S4
  } // sub2                             is declared in S1
  function sub3(){                      is passed in S3
    var x;
    x = 3;
    sub4(sub2);     3: passed in
  } // sub3
  function sub4(subx) {
    var x;
    x = 1;
    subx();         1:called by
  } // sub4
  x = 2;
  sub3();
} // sub1
```

❖ **ÖZET;**

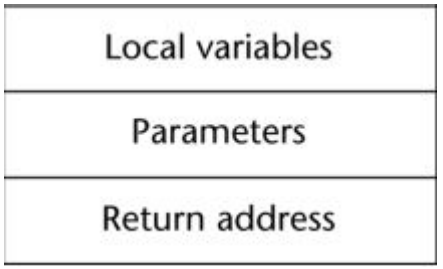
- ✓ Bir alt program tanımı, alt program tarafından temsil edilen eylemleri açıklar.
- ✓ Alt programlar, fonksiyonlar veya prosedürler olabilir.
- ✓ Alt programlardaki yerel değişkenler yığın dinamik veya statik olabilir.
- ✓ Üç parametre geçirme modeli: in mode, out mode & in-out mode.

- ✓ Bazı diller operatörün aşırı yüklenmesine izin veriyor
- ✓ Alt programlar kapsamlı (generic olabilir.
- ✓ Bir coroutine, çoklu girişleri olan özel bir alt programdır.

3) Ders09 - Implementing Subprograms:

❖ Implementing "Simple" Subprograms;

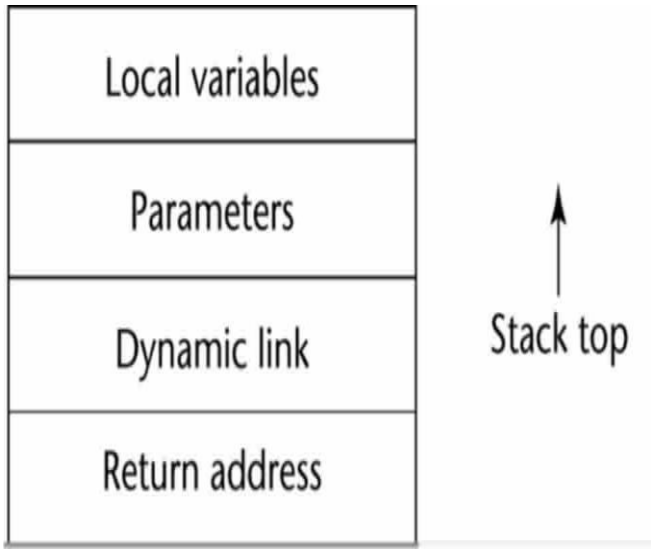
- ✓ Basit alt programlara sahip diller özyinelemeyi desteklemediğinden, belirli bir alt programın etkin bir versiyonunu bir seferde yalnızca bir tane olabilir.
- ✓ Bu nedenle, bir alt program için aktivasyon kaydının sadece tek bir örneği olabilir.
- ✓ Basit bir alt programın aktivasyon kaydı örneği sabit bir boyuta sahip olduğundan, statik olarak tahsis edilebilir. Ayrıca kod bölümüne de eklenebilir.



- ✓ Kodun ARI'lere eklenebileceğini unutmayın!! Ayrıca, dört program birimi farklı zamanlarda derlenebilir. Linker, ana program için çağrıldığında derlenen parçaları bir araya getirir.

❖ Stack-Dynamic Local Variables;

- ✓ ***Return address:*** Nereden çağrıldığını işaret eden adres. ***Dinamic link:*** Arayanın aktivasyon kayıt örneğinin tepesine gösterici.
- ✓ Statik kapsamlı dillerde, bu bağlantı, bir çalışma zamanı hatası oluştuğunda geri izleme bilgisi sağlamak için kullanılır.
- ✓ Dinamik kapsamlı dillerde, dinamik olmayan bağlantı yerel olmayan değişkenlere erişmek için kullanılır.



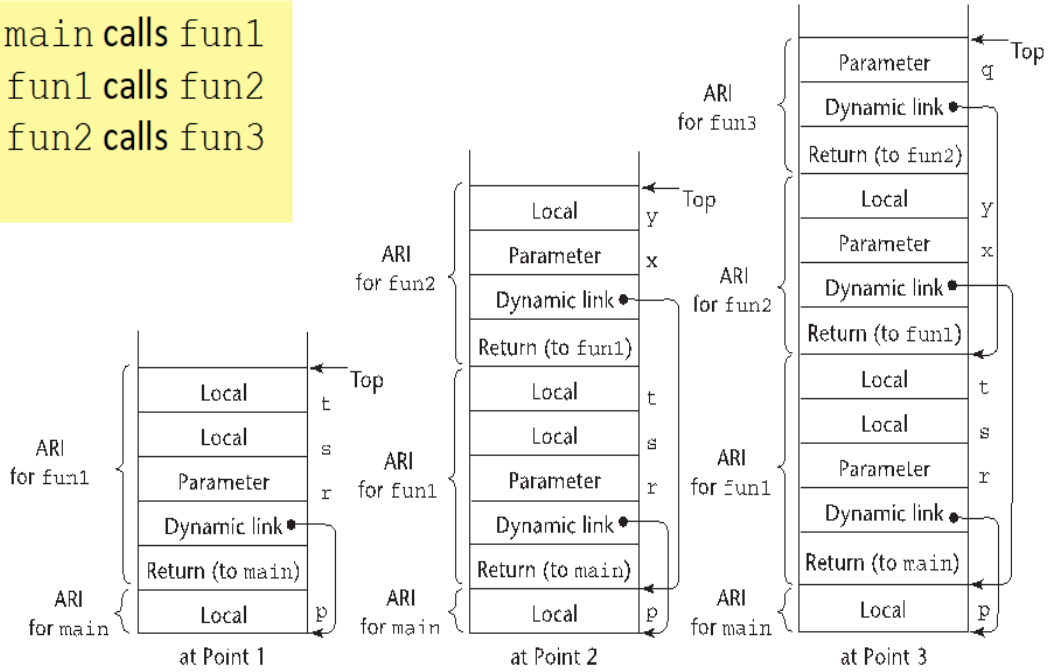
An Example Without Recursion

```
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 calls fun1
fun1 calls fun2
fun2 calls fun3

An Example Without Recursion

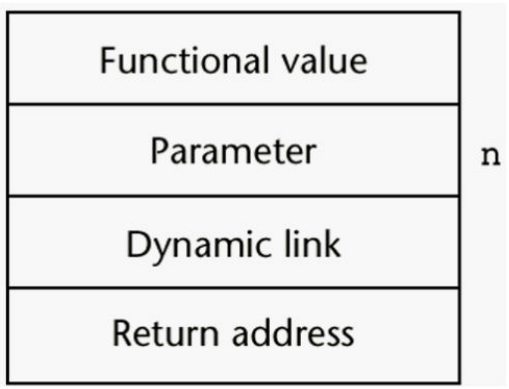
main calls fun1
fun1 calls fun2
fun2 calls fun3



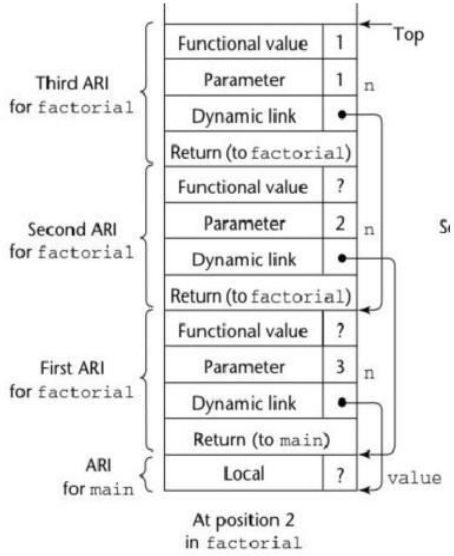
Recursion

```
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
}
```

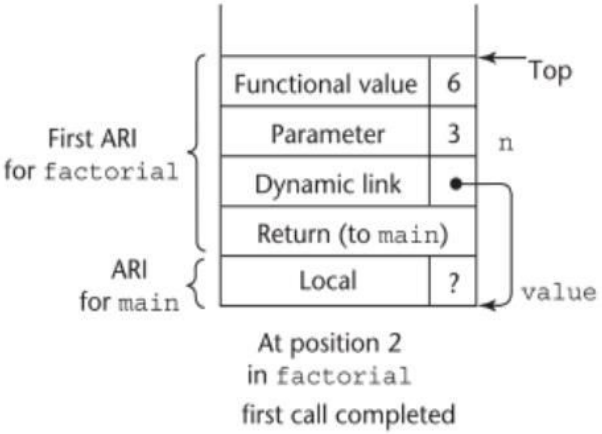
Activation Record for `factorial`



Stack contents during execution of `main` and `factorial`



Stack contents during execution of `main` and `factorial`



❖ **Nested Subprograms (İç İçe Alt Programlar);**

- ✓ ***Statik zincir (static chain),*** belirli aktivasyon kaydı örneklerini bağlayan statik bağlantıların bir zinciridir.
- ✓ Alt program A için bir aktivasyon kayıt örneğindeki ***statik bağlantı (static link),*** A statik ebeveyni aktivasyon kaydı örneklerinden birinin altına işaret eder.
- ✓ Bir aktivasyon kaydı örneğindeki statik zincir, onu statik atalarının tümüne bağlar.
- ✓ ***Static_depth,*** değeri iç içe geçme derinliği => olan statik bir kapsamla ilişkilendirilen bir tamsayıdır.

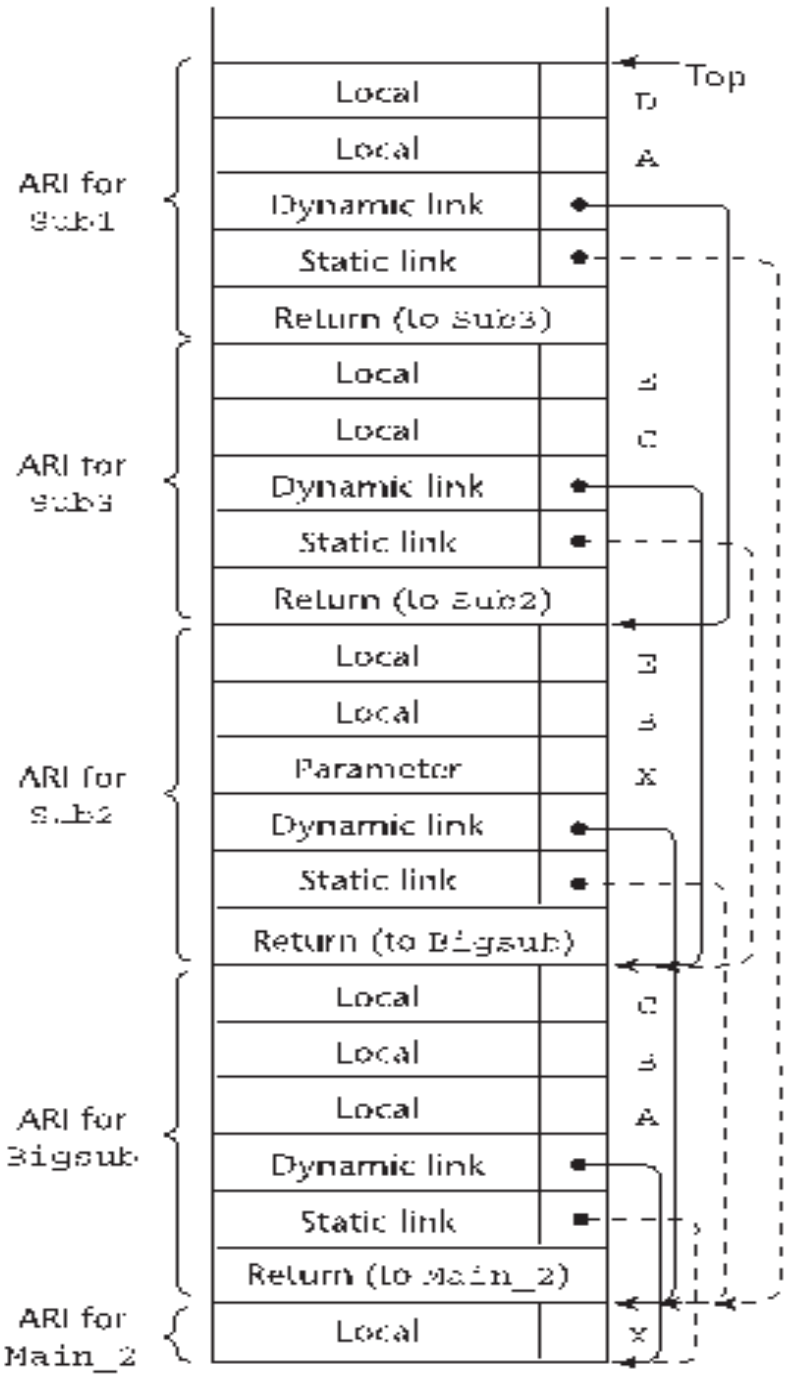
Example Ada Program*

```
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
  Bigsub;
```

Example Ada Program (continued)

- Call sequence for `Main_2`

  `Main_2` calls `Bigsub`
  `Bigsub` calls `Sub2`
  `Sub2` calls `Sub3`
  `Sub3` calls `Sub1`



❖ **Implementing Blocks;**

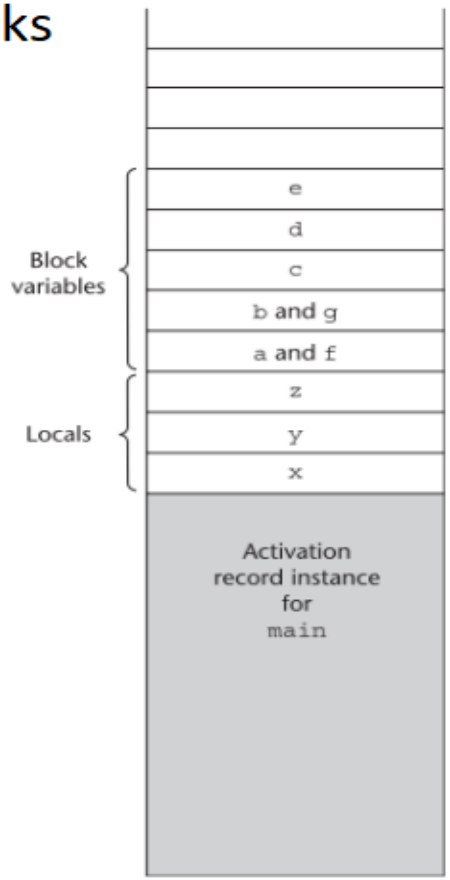
✓ Bloklara, her zaman aynı konumdan çağrılan parametresiz alt programlar olarak davranın. Her bloğun bir aktivasyon kaydı vardır; blok her çalıştırıldığında bir örnek oluşturulur.

✓ Bir blok için gereken maksimum depolama statik olarak belirlenebildiğinden, bu miktardaki alan aktivasyon kaydındaki yerel değişkenlerden sonra tahsis edilebilir.

Implementing Blocks

- Block variable storage when blocks are not treated as parameterless procedures

```
void main() {
  int x, y, z;
  while ( . . . ) {
    int a, b, c;
    . . .
    while ( . . . ) {
      int d, e;
      . . .
    }
  }
  while ( . . . ) {
    int f, g;
    . . .
  }
  . . .
}
```

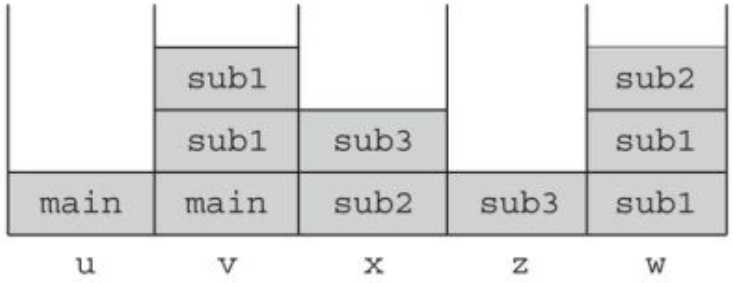


✓ **Derin Erişim (Deep Access):** yerel olmayan referanslar, dinamik zincirdeki aktivasyon kaydı örnekleri aranarak bulunur. Zincirin uzunluğu statik olarak belirlenemez. Her aktivasyon kaydı örneği değişken adlarına sahip olmalıdır.

✓ **Sığ Erişim (Shallow Access):** Local'leri merkezi bir yere koy. Her değişken adı için bir yığın. Her değişken adı için bir girişi olan merkezi bir tablo.

✓ **Shallow Access:**

```
void sub3() {
  int x, z;
  x = u + v;
  …
}
void sub2() {
  int w, x;
  …
}
void sub1() {
  int v, w;
  …
}
void main() {
  int v, u;
  …
}
```



(The names in the stack cells indicate the program units of the variable declaration.)

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

4) Ders12 - Logical Programming:

❖ Mantık dilindeki programlar sembolik mantık biçiminde ifade edilir. Sonuç üretmek için mantıksal bir çıkarım işlemi kullanılır.

❖ Mantıksal bir ifade, doğru olabilir de olmayabilir de. Mantıksal ifadeler, Nesnelerden ve nesnelerin birbirleriyle ilişkilerinden oluşur.

❖ Formal mantığın temel ihtiyaçları için kullanılabilecek mantık:

- ➢ Önerileri ifade etme,
- ➢ Öneriler arasındaki ilişkileri ifade etme,
- ➢ Yeni önermelerin diğer önermelerden nasıl çıkarılabileceğini açıklama.

❖ Önerilerdeki nesneler basit terimlerle temsil edilir: sabitler (constants) veya değişkenler (variables).

❖ ***Sabit (Constant):*** bir nesneyi temsil eden bir sembol.

❖ ***Değişken (Variable):*** farklı zamanlarda farklı nesneleri temsil edebilen bir sembol. Zorunlu dillerdeki değişkenlerden farklı.

❖ **Compound Terms (Bileşik Terimler);**

➢ İki bölümden oluşan bileşik terim,

✓ ***Functor:*** ilişkiyi adlandıran işlev simgesi.

✓ ***Tuple (Demet):*** Sıralı parametrelerin listesi.

✓ Örneğin >>> student(john), like(seth, OSX), like(nick, windows), like(jim, linux).

❖ **Logical Operators;**

Name	Symbol	Example	Meaning
negation	¬	¬ a	not a
conjunction	∩	a ∩ b	a and b
disjunction	∪	a ∪ b	a or b
equivalence	≡	a ≡ b	a is equivalent to b
implication	⊃ ⊂	a ⊃ b a ⊂ b	a implies b b implies a

Quantifiers

Name	Example	Meaning
universal	∀X.P	For all X, P is true
existential	∃X.P	There exists a value of X such that P is true

Booleans

No	false value
fail	false value
not(4)	logical not
; / ,	logical or / and (short circuit)
true	true value
Yes	true value

❖ **Sebep Formu (Clausal Form);**

➢ Aynı şeyi ifade etmenin çok fazla yolu vardır. Mesela, öneriler için standart bir form kullanmak gibi.

➢ Sebep formu:

➢ $B_1 \cup B_2 \cup \ldots \cup B_n \subset A_1 \cap A_2 \cap \ldots \cap A_m$ Tüm A'lar doğru ise, en az bir B doğru demektir.

➢ Öncül (Antecedent): sağ taraf, Sonuç (Consequent): sol taraf.

❖ **Terms - Variables & Structures;**

➢ Değişken (Variable): büyük harfle başlayan herhangi bir harf, rakam ve alt çizgi dizisi.

- Örnekleme (Instantiation): bir değişkenin bir değere bağlanması. Sadece bir tam hedefi yerine getirmek için sürdüğü sürece devam eder.
- Yapı (Structure): atomik önermeyi temsil eder. functor (parametre listesi).

❖ **Örnekler;**

- parent(X,Y) :- mother(X,Y). >>> "X, Y'nin annesi ise X, Y'nin ebeveynidir."
- grandparent(X,Z):- parent(X,Y), parent(Y,Z). >>> "Y, Z'nin ebeveyni ve X'de Y'nin ebeveyni ise X, Z'nin atasıdır."

Example

```
speed(ford,100).
speed(chevy,105).
speed(dodge,95).
speed(volvo,80).
time(ford,20).
time(chevy,21).
time(dodge,24).
time(volvo,24).
distance(X,Y) :-   speed(X,Speed),
                   time(X,Time),
                   Y is Speed * Time.
```

A query: `distance(chevy, Chevy_Distance).`

Example

```
likes(jake,chocolate).
likes(jake, apricots).
likes(darcie, licorice).
likes(darcie, apricots).

trace.
likes(jake, X), likes(darcie, X).
((1)) 1 Call: likes(jake, _0)?
(1) 1 Exit: likes(jake, chocolate)
(2) 1 Call: likes(darcie, chocolate)?
(2) 1 Fail: likes(darcie, chocolate)
((1)) 1 Redo: likes(jake, _0)?
(1) 1 Exit: likes(jake, apricots)
(3) 1 Call: likes(darcie, apricots)?
(3) 1 Exit: likes(darcie, apricots)
X = apricots
```