9. Alt Programlar

Bir programlama dilinde 2 temel soyutlama imkanı vardır: süreç soyutlama ve veri soyutlama. Yüksek Seviyeli programlama dileerinin ilk zamanlarında sadece süreç soyutlama vardı. Tüm programlama dillerinin merkezinde alt programlarla süreç soyutlama bulunmaktaydı. 1980' lerde çoğu kişi veri soyutlamanın da en az süreç soyutlama kadar önemli olduğunu düşünmeye başladı.

9.1. Alt Programların Temelleri
9.1.1. Genel Alt Program Özellikleri

Bu ünitenin ilerki kısımlarında bahsedeceğimiz coroutineler haric tüm alt programlar şu özelliklere sahiptir:

- Her alt programın tek bir giriş noktası vardır.
- Alt programı çağıran program, alt programın çalışması tamamlanıncaya kadar askıya alınır. Bu da herhangi bir zamanda sadece bir alt programın çalışabileceği anlamına gelir.
- Alt programın çalışması bittiğinde kontrol daima çağıran yere döner.

Bunlara alternatif olarak cocoutine'ler eş zamanlı birimler (concurrent units) vardır.

9.1.2. Temel Tanımlar

Alt program, kendi bünyesinde yapılan işlemlerin soyutlaması için bir arayüzdür. **Alt program çağrısı**, alt programın işletilmesi için yapılan açık taleptir. Bir alt program çağrılmışsa ancak henüz çalışması bitmediyse **aktif**tir.

Bir **alt programın başlığı**, yani tanımının ilk kısmı, birçok bilgi sağlar. Öncelikle alt program **türü**nü (fonksiyon ya da prosedür) ifade eder. Eğer bir dilde birden fazla türde alt program varsa alt programın türü bir özel kelimeyle ifade edilir. Eğer altprogram anonim değilse başlıkta **adı** da bulunur. Eğer kullanılıyorsa **parametre listesi** de başlıkta belirtilir.

Örn. Python'da fonksiyon **def** deyimi ile tanımlanır. (Ruby'de de def kullanılır.)

def adder (parameters):

JavaScript'te tanım **function** deyimi ile başlar.

C' de,
void adder(parameters) tanımında void deyimi bu alt programın geri değer döndürmeyeceğini ifade eder.

Python'daki **def** deyiminin çalıştırılabilir bir deyim olması onu diğer programlama dillerinden ayırır. Def deyimi çalıştırıldığında verilen adı fonksiyonun gövdesi ile eşler. Bir fonksiyonun def'i çalıştırılmadan o fonksiyon çağrılamaz.

Ruby'deki metotlar diğer programlama dillerinin alt programlarından birçok yönden ayrılırlar. Ruby'de metot tanımları sınıf tanımı içinde veya dışında yapılabilir. Eğer sınıf dışında tanımlanırlarsa kök nesneye yani Object sınıfına bağlanırlar.

Tüm Lua fonksiyonları anonimdir ancak isimleri varmış gibi tanımlanabilirler.
Örn. function cube(x) return x*x*x end
cube = function (x) return x*x*x end

Bir alt programın **parametre profili**, parametrelerinin sayısı, tipi ve sırasını içerir. Bir alt programın **protokolü**, parametre profili ve eğer fonksiyonsa dönüş tipinden oluşur.

Alt programların tanımları olduğu gibi açıklamaları da olabilir. C dilinde buna fonksiyon prototipi denmektedir. Fonksiyon prototipinde altprogram protokolü belirtilir; fonksiyon gövdesi yazılmaz. Bu açıklamalar ileri referansa izin vermeyen dillerde önemlidir. C ve C++'tan başka çoğu dilde altprogramlar için açıklama yazmak gerekmez çünkü alt programların çağrıldığı yerin üzerinde biryerde tanımlanma zorunluluğu yoktur.

9.1.3. Parametreler

Alt programların hesaplama yapmak için kullanacağı değerler ya başka bir yerde tanımlanmış olan ama alt programa görünür olan değişkenler ya da alt programa parametrelerle geçirilen değerlerdir. Parametre olarak geçirilen veri o alt programın yerel değişkenleri olurlar ve onlara adları ile erişilir. Parametre geçişi, yerel olmayan değişkenlere doğrudan erişmekten daha esnek bir yoldur. Yerel olmayan değişkenlere fazla erişim güvenilirliği azaltmaktadır.

Alt program başlığındaki parametrelere **formal parametreler** denir. Çoğunlukla belleğe alt program çağrıldığında bağlandıkları için sözde değişkenler olarak düşünülürler.

Alt program çağrı ifadeleri, alt programın adı ve alt programın formal parametre listesine bağlanacak olan parametre listesini içerir. Bu parametrelere **gerçek (actual) parametreler** diyoruz.Bunlar formal parametrelerden ayrı tutulmalıdır çünkü farklı kısıtlara ve kullanımlara sahiptirler.

Neredeyse tüm programlama dillerinde gerçek ve formal parametreler arasındaki bağ pozisyona göre yapılır: ilk gerçek parametre ile ilk formal parametre ... gibi. Böyle parametrelere konumsal parametreler denir ve bu yöntem çok uzun olmayan parametre listeleri için gerçek parametreleri formal parametrelere bağlamanın etkin ve güvenli bir yoludur. Parametre listesi çok uzun olduğunda programcı hata yapabilir. Bazı dillerde buna çözüm olarak **anahtar kelime parametreleri** bulunmaktadır. Örneğin Python fonksiyonları bu şekilde çağrılabilir.
Örn: sumer(length = my_length,
list = my_array,
sum = my_sum)

burada length, list ve sum formal, my_length, my_array ve my_sum da gerçek parametrelerdir.

Anahtar kelime parametreler, gerçek parametre listesinde herhangi bir yerde görülebilirler, konumları önem taşımaz. Bu durum bir avantajdır. Ancak bu durumda da kullanıcı alt programın bütün formal parametrelerinin adını bilmek zorundadır. Bu da anahtar kelime parametrelerinin dezavantajıdır.

Anahtar kelime parametrelere ek olarak Ada, Fortran 95+ ve Python'da konumsal parametreler bulunur. Bir altprogram çağrısında karışık olarak kullanılabilirler:

```
sumer( my_length,
          sum = my_sum,
           list = my_array )
```

Bu kullanımın kısıtı, bir kere parametre listesinde anahtar kelime parametre görüldüğünde, ondan sonraki tüm parametrelerin de anahtar kelime olması zorunluluğudur. Çünkü anahtar kelime parametre kullanımından sonra konumlar bozulmuş olabilir.

Python, Ruby, C++, Fortran 95+, Ada ve PHP'de formal parametrelerin varsayılan değerleri olabilmektedir. Eğer fonksiyon çağrısında o parametre için bir değer geçilrimezse varsayılan değer kullanılır.

Örn: Python

```
    def compute_pay(income, exemptions = 1, tax_rate)
```

Python çağrısında kullanılmayan parametre yerine virgül konmayabilir. Virgül kullanımı bir sonraki parametrenin konumunu belirtmek içindir. Anahtar kelime parametre desteği olduğunda bu gerekli değildir. Virgül kullanılmadığında adı geçmeyen parametreden sonraki tüm parametrelercanahtar kelime Parametre olmalıdır. Örneğin, bir çağrı şu şekilde yapılabilir:

```
    pay = compute_pay(20000.0, tax_rate = 0.15)
```

C++' ta anahtar kelime parametre kullanımı yoktur o yüzden varsayılan değeri olan parametre kullanımı biraz değişmektedir. Varsayılan değeri olan parametre den sonraki tüm parametrelerin varsayılan değerleri olmalıdır. C++'da örnek bir fonksiyon başlığı şu şekilde yazılabilir:

```
    float compute_pay(float income, float tax_rate, int exemptions = 1)
```

Bu fonksiyona yapılacak örnek bir çağrı:

```
    pay = compute_pay(20000.0, 0.15);
```

Formal parametrelerin varsayılan değerlerinin desteklenmediği dillerde formal parametre sayısı ile fonksiyon çağrısındaki gerçek parametre sayısı birbirine uymak zorundadır. Fakat bu durumun dışında kalan diller vardır: c, c++, Perl, JavaScript ve Lua. Bu durumda, yani formal parametre sayısından daha az gerçek parametre kullanıldığında parametrelerin doğru eşleşmesinin sağlanması programcıya bırakılmıştır. Bunun hatalara açık bir tasarım olduğu açıktır ancak bazen de bu kullanım uygundur. Örneğin C'deki printf fonksiyonu

istenen sayıda veriyi ekrana basabilir. C#'ta hepsinin tipi aynı olmak şartıyla değişken sayıda parametre kullanımı desteklenir.

```
public void DisplayList(params int[] list) {
        foreach (int next in list) {
                Console.WriteLine("Next value {0}", next);
        }
}
```

DisplayList'in MyClass sınıfı için tanmlandığını düşünürsek,

```
Myclass myObject = new Myclass;
int[] myList = new int[6] {2, 4, 6, 8, 10, 12};
```

DisplayList şu iki şekilden biri ile çağrılabilir:

```
myObject.DisplayList(myList);
myObject.DisplayList(2, 4, 3 * x - 1, 17);
```

9.1.4. Prosedürler ve Fonksiyonlar

Alt programların iki ayrı türü vardır: Fonksiyonlar değer döndürürler ve prosedürler değer döndürmezler. Çoğu dilde prosedür fonksiyondan ayrı bir biçime sahip değildir. Fonksiyonlar değer döndürmeyecek şekilde tanımlanır ve prosedür olarak kullanılabilirler. Ada dilindeki prosedürler **procedure**, Fortran'da da **subroutine** olarak adlandırılırlar. Çoğu diğer dilde prosedür desteği yoktur. Prosedürler iki şekilde değer üretebilirler:

1. Formal parametre olmayan ama hem prosedüre hem de çağıran program birimine görünür olan değişkenlerin değerini değiştirerek,
2. Formal parametrelerin çağıran yere veri transfer etmesine izin verilen durumlarda o parametrelerin değerlerini değiştirerek.

Fonksiyonlar yapısal olarak prosedürlere benzerler ama anlamsal olarak matematik fonksiyonları üzerine modellenmişlerdir. Eğer fonksiyon modeli aslına uygun ise yan etki yaratmaz (kendi parametreleri veya fonksiyon dışında tanımlanmış herhangi bir değeri değiştirmez). Böyle saf bir fonksiyon sadece değer döndürür. Ancak pratikte, çoğu programlama dilinde fonksiyonlar yan etkiye sahiptir.

Fonksiyonlar bir ifade içinde adları ve parametreleri ile çağrılırlar. Fonksiyonun çalışması ile üretilen değer çağrıldığı yere döner.Yan etkisi olmayan bir fonksiyonun yarattığı tek etki döndürdüğü değerdir.

9.2. Yerel Referans Ortamları

9.2.1. Yerel Değişkenler

Alt programlar kendi değişkenlerini tanımlayabilirler. Kapsamları alt programın gövdesi olduğundan bunlara yerel değişken denir. Yerel değişken statik ve stack-dinamik olabilirler. Eğer yerel değişkenler stack-dinamik ise alt program çalışmaya başladığında belleğe bağlanırlar ve alt programın çalışması sonlanınca da bellekle bağları kesilir.

Stack-dinamik değişken kullanımının birçok avantajı vardır. İlki alt programa

esneklik sağlamalarıdır. Çünkü bu tip değişkenler özyinelemeli alt programların gereğidir. Stack-dinamik yerel değişkenler aktifken, diğer aktif olmayan altprogram yerel değişkenleriyle bellek paylaşımı yapabilir. Bellek kapasitesi düşük olan bilgisayarlar için bu bir avantajdır.

Stack-dinamik değişkenlerin dezavantaşları da şöyle sıralanabilir:
1) Bu değişkenler için bellek tahsisi, ilkleme ve bellekten ayrılma işlemleri her alt program çağrısında gerçekleştiğinden zaman açısından maliyetlidir.
2) Bu değişkenlere erişim dolaylı yoldan sağlanır ancak statik değişkenlere erişim soğrudan sağlanır. Yerel değişkenlerin bellekte nereye yerleşeceği çalışma zamanında kararlaştırılır.
3) Tüm yerel değişkenler stack-dinamik olsaydı programlar tarihçe tutamazdı.

Çoğu modern programlama dilinde alt programların yerel değişkenleri varsayılan olarak stack-dinamiktir. C ve C++'ta *static* kelimesi ile statik yerel değişken tanımı yapılabilir.

```
int adder(int list[], int listlen) {
        static int sum = 0;
        int count;
        for (count = 0; count < listlen; count ++)
        sum += list [count];
        return sum;
}
```

Python'da bir global değişkene bir fonksiyon içinde değer atanamaz. Eğer atanmışsa, o değişken global değil, aynı isimli yerel bir değişken olarak işlem görür ve globalin değeri etkilenmez.

9.2.2. İç içe alt programlar

Bu fikir, hem mantık hem de kapsamda bir hiyerarşi yaratmak amacıyla, ilk defa Algol 60 ile ortaya çıkmıştır. Böylece kapsayan fonksiyonun yerellerine erişim de sağlanabilmiştir. C, iç içe alt program tanımlamaya izin vermez. Yakın zamanda JavaScript, Python, Ruby, Lua ve çoğu fonksiyonel dillerde kullanma izin verilmiştir.

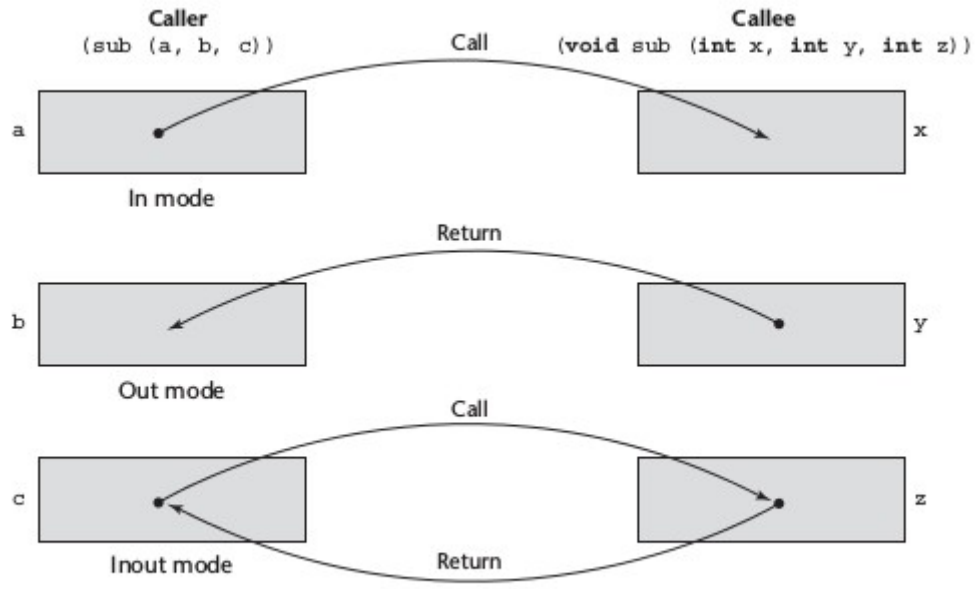
9.3. Parametre Geçiş Yöntemleri
9.3.1. Parametre Geçişinin Anlamsallığı

Formal parametreler 3 ayrı anlamsal model ile sınıflandırılabilir:
1) In mode – gerçek parametrelerden veri alma
2) Out mode – gerçek parametrelere değer gönderme
3) Inout mode – yukarıdakilerin ikisini de gerçekleştirme

Örneğin iki tane integer diziyi (list1, list2) parametre olarak alan bir fonksiyon düşünelim. Bu alt program list1' i list2' ye ekleyerek sonucu list2' nin yeni versiyonu olarak döndürsün. Yani alt program list1 ve list2' den yeni bir dizi üretmeli ve onu döndürmeli bu arada list2' yi de değiştirmeli. Bu alt program için list1 inmode olmalı çünkü bu dizinin değerine alt program ihtiyaç duyuyor ve alt programın çalışması bitince list1' in değerinde değişme olmayacak. List2 ise inout mode olmalıdır, çünkü hem alt programa değeri gerekiyor hem

de yeni değeri olacak. 3. dizi ise out mode olmalı, çünkü ilk değeri yok.

Parametre geçişinde iki kavramsal model vardır: ya gerçek değer kopyalanır ya da bir erişim yolu iletilir. Erişim yolu çoğu zaman bir işaretçi veya referanstır. Şekil 1' de değerlerin kopyalandığı durumda parametre geçişi için üç anlamsal model gösterilmiştir.



Şekil 1. Parametre geçişinin üç anlamsal modeli (değerlerin kopyalandığı durumda)

9.3.1.1. Pass-By-Value (Değerle geçirme)

Eğer parametreler değer ile geçirilirse (pass-by-value) gerçek parametrelerin değeri ile ilgili parametre ilklenir ve sonra formal parametre, alt programın yerel değişkeni gibi davranır. İn mode yönteminin uygulanması bu şekilde olur. Pass-by-value genelde kopyalama ile gerçekleşir. Bazen bir erişim yolu da parametre olarak geçirilebilir ancak bu durumda parametrenin yazmaya korumalı olması gerekir.

Pass-by-value' nin avantajı hem erişimin hem de bağlamanın hızlı olmasıdır. Dezavantajı ise parametreye değer kopyalamak için ek bellek kullanımına ihtiyaç duyulmasıdır.

9.3.1.2. Pass-By-Result (Sonuçla geçirme)

Sonuç ile geçirmede (pass-by-result), alt programa hiçbir değer geçirilmez. İlgili formal parametre yerel değişken gibi davranır ancak kontrol çağrıyı yapana geri dönmeden önce değeri çağıranın gerçek parametresine aktarılır.

Pass-by-result da pass-by-value'nin sahip olduğu avantaj ve dezavantajlara sahiptir ancak başka dezavantajları da vardır. Örneğin gerçek parametre

çakışması olabilir. Sub1(p1,p1) çağrısının yapıldığını düşünelim. Sub1 fonksiyonunda da farklı isimlerde iki formal parametre olsun. Bu formal değerlerden en son hangisi geri kopyalanırsa p1'in değeri o olacaktır. Böylece gerçek parametrelerin kopyalanma sırası değerlerini belirlemiş olur.

Aşağıda verilen C# kodunu inceleyelim. Burada out belirteci ile formal parametreler pass-by-result ile geçirilirler. Fonksiyonun çalışması bittikten sonra gerçek parametreye önce x kopyalanırsa a= 35, önce y kopyalanırsa a = 17 olacaktır.

```
void Fixer(out int x, out int y) {
        x = 17;
        y = 35;
}
. . .
f.Fixer(out a, out a);
```

Pass-by-result kullanımı ile oluşabilecek başka bir problem de parametrelerin belleğe ne zaman kopyalanacağıdır. Kopyalama, çağrı zamanında veya dönüş zamanında yapılabilir. Bu durum fonksiyonların çalışmaları sonucu gerçek parametrelerin farklı değerlere sahip olmasına sebep olabilir. Böyle olduğunda out-mode parametrelere adres bağlamayı alt programın girişinde yapan bir dile ait kod, bu işlemi dönüş zamanında yapan bir dile taşınamaz.

9.3.1.3. Pass-By-Value-Result (Değer ve sonuçla geçirme)

Inout mode parametreler için gerçekleştirim yöntemlerinden biridir. Pass-by-value ve pass-by-result' un kombinasyonudur. Gerçek parametrelerin değerleri formal parametreleri ilklemek için kullanılır ve daha sonra formal parametreler yerel değişkenler olarak kullanılır. Alt program sonlandığında formal parametre değerleri gerçek parametrelere kopyalanır. Bu yönteme pass-by-copy de denir.

Pass-by-value ve pass-by-result modellerinin parametreler için gereken ek bellek ve kopyalama için gereken zaman ve pass-by-result' un gerçek parametrelere kopyalama sırası ile ilgili problemlerini bu modelde görmekteyiz.

9.3.1.4. Pass-By-Reference (Referansla geçirme)

Inout mode parametreler için diğer bir gerçekleştirim yöntemidir. Değerleri ileri veya geri kopyalamaktan ziyade pass-by-reference ile alt programa genellikle adres olan bir erişim yolu gönderilir. Bu, gerçek parametrenin bellek hücresine erişim yolu sağlar. Böylece alt programın çağrıyı yapan program birimindeki gerçek parametreye doğrudan erişimine izin verilir.

Pass-by-reference' nin avantajı, hem zaman hem de bellek kullanımı açısından etkin oluşudur. Kopyalama yapılmadığı için kopyalamada ihtiyaç duyulan bellek ihtiyacı burada yoktur.

Fakat pass-by-reference yönteminin birçok dezavantajı vardır. Öncelikle, formal parametrelere erişim, dolaylı adresleme sebebiyle, pass-by-value' den daha yavaş olur. İkincisi, eğer alt programla sadece tek yönlü iletişim kurulması gerekiyorsa, hatalı veya yanlışlıkla yapılan değişiklikler gerçek parametreyi

değiştireceğinden tehlikelidir. Pass-by-reference ile ilgili diğer bir sorun da alias oluşmasıdır ve bu okunabilirlik ve güvenilirlik açısından istenmeyen bir durumdur.

Örnek: C++ kodu (gerçek parametre çakışması)

- void fun(int &first, int &second)

eğer çağrıyı fun(total, total) şeklinde yaparsak first ve second parametreleri alias olur.

- fun(list[i], list[j]) şeklinde çağrı yapılırken eğer i=j ise bu iki parametre alias olur.
- Eğer dizi elemanı ve dizi adı referansla gönderilirse fun(list[i],list) çağrısı da alias oluşumuna sebep olabilir çünkü dizinin adı ile tüm elemanlarına ulaşabiliriz.
- Global değişkenle alias oluşumu

```
int *global;
void main(){
    …
    sub(global);
}
void sub(int *param){ //param, global ile alias oluyor
        …
}
```

9.3.1.5. Pass-By-Name

Inout mode parametre geçişlerinden biridir. Parametre isimle geçirildiğinde gerçek parametre ilgili formal parametre ile değiştirilir. Daha önceki metotlarda formal parametreler gerçek parametrelere veya adreslere, alt program çağrısı yapıldığında bağlanmaktadır. Bir pass-by-name formal parametresi, alt program çağrıldığında bir erişim yöntemine bağlanır. Bir değere veya adrese bağlanması bir atama işleminde veya referansta kullanıldığında gerçekleşir. Bu tip parametreler kullanışsız ve karmaşıktır ve programa ek yük getirirler. Okunabilirliği ve güvenilirliği azaltırlar.

9.3.2. Parametre Geçirme Yöntemlerinin Gerçekleştirimi

Çoğu modern dilde parametre iletişimi run-time stack aracılığıyla yapılır. Programların çalışmasını yöneten run-time sistem, run-time stack' in ilklenmesi ve sürdürülmesi ile ilgilenir. Run-time stack, alt program kontrol bağlama ve parametre geçişi için kullanılır.

Pass-by-value parametrelerin değerleri stack' ta biryere kopyalanır. Bu yerler daha sonra ilgili formal parametrelerin depoları gibi kullanılır. Pass-by-result da bunun tam tersi şekilde çalışır. Stack' ta pass-by-result gerçek parametrelerinin değerleri bulunur ve alt programın çalışması bittiğinde çağıran program birimi bu değerleri stack' tan alabilir. Pass-by-value-result da bu ikisinin birleşimi şeklinde çalışır.

Pass-by-reference parametrelerin gerçekleştirimi içlerinde en kolay olanıdır. Gerçek parametrenin tipinden bağımsız olarak sadece adresi stack' a yerleştirilir.

9.3.3. Bazı Dillerde Parametre Geçiş Yöntemleri

C' de pass-by-value kullanılır. Pass-by-reference için parametre olarak işaretçiler kullanılır. C ve C++' ta işaretçi parametrenin başına **const** tanımlayıcısı da eklenebilir. Bu durumda parametreler yazmaya korumalı olur.

C++, referans tipi denen özel bir tişaretçi tipine sahiptir. Genelde parametre olarak kullanılır. Const olarak tanımlanabilirler.

```
void fun(const int &p1, int p2, int &p3) { . . . }
```

Const parametrelerle in mode parametreler aynı değildir. Const parametrelere kesinlikle değer atanamıyor ancak in mode parametrelere değer atayabiliriz ancak değişiklik gerçekleşmez.

C ve C++ gibi Java da pass-by-value kullanır. Fakat nesne parametreler etkisel olarak pass-by-reference gibidirler. Nesneler sadece referanslar aracılığıyla erişilebilirler. Nesne referansının kendisi değiştirilemez ancak referans ettiği nesne alt program tarafından değiştirilebiliyorsa o zaman değiştirilebilir. Çünkü skaler değişkenler referans değişkenler tarafından doğrudan işaret edilemez ve Java'da işaretçi kullanımı da yoktur. Dolayısıyla skaler değişkenler referans ile geçirilemezler.

Ada ve Fortran95+'ta her formal parametre için farklı mod belirtebiliriz.

C#'ta varsayılan pass-by-value ile parametre geçişidir. Referanslar kullanılarak hem çağrıda hem de formal tanımda pass-by-reference yapılabilir.

```
void sumer(ref int oldSum, int newOne) { . . . }
. . .
sumer(ref sum, newValue);
```

C#' ta out mode da desteklenir. Bunlar pass-by-reference parametrelerdir ancak ilk değere ihtiyaç duymazlar. Out tanımlayıcısı ile tanımları yapılır.

PHP' de de parametre geçişi C#' a benzer ancak hem actual hem de formal parametreler pass-by-reference'i belirtirler. & işareti ile gerçekleştirilir.

Python ve Ruby' deki parametre geçiş yöntemi pass-by-assignment olarak adlandırılır. Çünkü tüm veri değerleri nesnedir ve her değişken bir nesneye referanstır. Pass-by-assignment ile gerçek parametre değeri formal parametreye atanır. Dolayısıyla burada bir pass-by-reference etkisi oluşur. Çünkü tüm gerçek parametrelerin değerleri referanslardır.

Çoğu nesne değiştirilemez (immutable) dir. Saf bir nesne tabanlı programlama dilinde x = x+1 gibi bir ifade x ile referans edilen nesneyi değiştirmez. Daha ziyade x' in referans ettiği nesneyi alır, 1 arttırır, yeni değerle yeni bir nesne

yaratır ve x' in yeni nesneyi referans etmesi sağlanır.

9.3.4. Parametre Geçiş Örnekleri

C dilinde,

```
void swap1(int a, int b) {
        int temp = a;
        a = b;
        b = temp;
}
```

Bu fonksiyonun swap1(c, d); ile çağrıldığını varsayalım. (pass-by-value) yapılan işlemler kabaca şöyledir:

```
a = c         ->Move first parameter value in
b = d         ->Move second parameter value in
temp = a
a = b
b = temp
```

```
void swap2(int *a, int *b) {
        int temp = *a;
        *a = *b;
        *b = temp;
}
```

Bu fonksiyonun swap2(&c, &d); ile çağrıldığını varsayalım. işlemler:

```
a = &c        ->Move first parameter address in
b = &d        ->Move second parameter address in
temp = *a
*a = *b
*b = temp
```

C++ ile swap2

```
void swap2(int &a, int &b) {
        int temp = a;
        a = b;
        b = temp;
}
```

Bu basit işlem Java ile yapılamaz çünkü Java' da işaretçi ve C++' taki gibi referanslar bulunmaz. Referanslar Java' da sadece bir nesneyi işaret edebilirler.

Ada in out mode için pass-by-value-result kullanır.

```
procedure swap3(a : in out Integer, b : in out Integer) is
        temp : Integer;
        begin
        temp := a;
        a := b;
        b := temp;
        end swap3;
```

Çağrı swap3(c, d);

işlemler:

```
addr_c = &c        ->Move first parameter address in
addr_d = &d        ->Move second parameter address in
a = *addr_c        ->Move first parameter value in
b = *addr_d        ->Move second parameter value in
temp = a
a = b
b = temp
*addr_c = a        ->Move first parameter value out
*addr_d = b        ->Move second parameter value out
```

C-benzeri sözdizimi

```
int i = 3; /* i is a global variable */
void fun(int a, int b) {
        i = b;
}
void main() {
        int list[10];
        list[i] = 5;
        fun(i, list[i]);
}
```

```
addr_i = &i                ->Move first parameter address in
addr_listi = &list[i]      ->Move second parameter address in
a = *addr_i                ->Move first parameter value in
b = *addr_listi            ->Move second parameter value in
i = b                      ->Set i to 5
*addr_i = a                ->Move first parameter value out
*addr_listi = b            ->Move second parameter value out
```

Eğer pass-by-reference kullanılırsa i ve a alias olur. Eğer pass-by-value-result kullanılırsa i ve a aliasa olmaz.

9.4. Parametre Olarak Alt Programlar

Programlamada alt programların başka alt programlara parametre olarak gönderilmeleri ile daha etkin şekilde çözülebilecek problemler vardır. En genel örnek matematik fonksiyonlarıdır. Bu basit bir iş gibi gözükse de detayları kafa karıştırıcı olabilir. Parametre olarak geçirilen fonksiyonun çalışması ile gerçekleşecek parametre tip kontrolü problem olabilir.

C ve C++' ta fonksiyonlar parametre olarak geçirilmez ancak fonksiyonlara işaretçiler geçirilebilir. Bu işaretçinin tipi fonksiyon protokolünde belirtilir. Protokol tüm parametre tiplerini içerdiği için tip kontrolü sorunu olmaz. Fortran95+' da parametre olarak geçirilen fonksiyonların tip kontrolü zorunludur ve bunu yapan mekanizma vardır.

Parametre olarak fonksiyon geçirebilen dillerdeki başka bir problem sadece iç içe fonksiyon tanımına izin veren dillerde görülür. Parametre olarak geçirilen fonksiyonun hangi referans ortamını kullanacağı problemi ortaya çıkmaktadır. 3 seçenek vardır:

1) Sığ bağlama (Shallow binding) – geçirilen alt programa çağrı yapan ifadenin bulunduğu ortam
2) Derin bağlama (Deep binding) – geçirilen alt programın tanımlandığı ortam.
3) Tasarsız bağlama (Ad-hoc binding) – alt programın gerçek parametre olarak geçirildiği ortam.

Örnek JavaScript kodu:

```
function sub1() {
	var x;
	function sub2() {
	alert(x); // Creates a dialog box with the value of x
};
function sub3() {
	var x;
	x = 3;
	sub4(sub2);
};
function sub4(subx) {
	var x;
	x = 4;
	subx();
};
x = 1;
sub3();
};
```

sub2' nin sub4' ten çağrıldığındaki işleyişine bakalım.

- Sığ bağlama için sub2' nin referans ortamı sub4' tür. Yani sub2' de kullanılan x, sub4' ün yerel değişkeni olan x' tir. Çıktı 4 olur.
- Derin bağlama için sub2'nin referans ortamı sub1' dir. X, sub1' in yerel değişkleni olan x'tir ve çıktı 1 olur.

- Tasarsız bağlama için sub2' nin referans ortamı sub3' tür ve çıktı da 3 olur.

Bazı durumlarda alt programın parametre olarak geçirildiği ve tanımlandığı ortam aynıdır. O zaman derin ve tasarsız bağlama için referans ortam aynı olur.

9.5. Alt Programları Dolaylı Çağırmak

Bu yöntem grafiksel kullanıcı arayüzü (GUI) kullanımlarında event handler' lar ile genellikle kullanılır. Bu yeni bir kullanım değildir. C ve C++' ta fonksiyona işaretçi tanımlama var, bu işaretçi aracılığıyla fonksiyon çağrılabiliyor.

C++' ta float (*pfun)(float, int); şeklinde tanımlanan fonksiyon işaretçisi, float ve int parametre alan ve float değer döndüren herhangi bir fonksiyona işaret edebilir.

```
int myfun2 (int, int); // fonksiyon tanımı
int (*pfun2)(int, int) = myfun2; // pointer yaratıldı ve myfun2'yi işaret etmesi
                                 // için  ilklendi
pfun2 = myfun2; // Fonksiyonun adresi işaretçiye atanıyor
```

myfun2 şu iki şekilde çağrılabilir:

```
(*pfun2)(first, second);
pfun2(first, second);
```

9.6. Aşırı Yüklenmiş Alt Programlar (Overloaded Sub Programs)

Başka bir altprogramla aynı isimle aynı referans ortamında tanımlanan fanksiyonlara aşırı yüklenmiş fonksiyonlar denir. Aynı ismi paylaşan fonksiyonların protokolleri (parametrelerinin tipleri, sıraları, sayıları veya dönüş tipi) farklı olmak zorundadır.

Tip zorlaması olan dillerde bir durum, C++ örneği:

```
int fun(int)
float fun(int) //bu iki fonksiyondan hangisinin çalıştırılacağına karar verilemez.
```

Varsayılan parametreleri olan aşırı yüklenmiş alt programlar da karışıklık yaratabilir. Örn: C++

```
void fun(float b = 0.0);
void fun();
. . .
fun(); // hangi fonksiyonun çağrılacağına karar verilemez ve derleme hatası alınır.
```

9.7. Generic Alt Programlar

Dört farklı türde veri tutan diziyi sıralama işlemi dört ayrı fonksiyon tarafından yapılmamalıdır. Bir polimorfik (çok biçimli) alt program, farklı tiplerde parametreler alarak farklı işlemlerde kullanır. Aşırı yüklenmiş alt programlar

tasarsız polimorfizm denen özel bir tür çok biçimliliğe sahiptirler.

Nesne tabanlı dillerde sub-type polimorfizm vardır. T tipinden bir değişken, T tipinden olan veya T tipinden türetilmiş her nesneye erişebilir.

Ruby ve Python' da daha genel bir polimorfizm kavramı vardır. Bu dillerde değişkenlerin olmadığı gibi, parametrelerin de tipi yoktur. Dolayısıyla bir metod herhangi tipte gerçek parametrelerle çağrılabilir. Parametrik polimorfizm, generic parametreler alan alt programlar tarafından sağlanır. Bu tip alt programlar genellikle **generic** olarak adlandırılır.

9.8. Fonksiyon Tasarım Konuları

- Yan etkilere izin verilecek mi
- Ne tipte değer döndürülebilir
- Kaç tane değer döndürülebilir

9.8.1. Yan etkiler

Yan etkileri önlemek için parametreler daima in mode olmalıdır. Örneğin Ada dilinde durum böyledir. Çoğu diğer dilde pass-by-reference veya pass-by-value-result ile fonksiyonların yan etki yaratması veya alias üretmes mümkündür. Haskell gibi saf fonksiyonel dillerde değişken olmadığından fonksiyonların yan etkileri de yoktur.

9.8.2. Döndürülen Değerlerin Tipleri

Çoğu buyurgan dilde dönüş tipleri kısıtlanmıştır. C' de dizi veya fonksiyon döndürülemez. Bu ikisini gerçekleştirmek için işaretçiler kullanılabilir.

Ada, Python, Ruby, Lua fonksiyonları her türde veriyi döndürürler. Sadece Ada'da fonksiyon döndürülmez ama fonksiyona işaretçi döner.

Java ve C#' ta fonksiyon yoktur metodlar vardır. Metodlar herhangi bir nesneyi döndürebilir. Metodlar kendileri bir tip olmadıkları için döndürülmezler.

9.8.3. Döndürülen Değer Sayısı

Çoğu dilde fonksiyonlardan tek bir değer döndürülür. Fakat her dilde durum böyle değildir. Ruby' de bir metod birden fazla değer döndürebilir. Ruby metodunda bir return deyimini takiben bir ifade yoksa nil döner. Eğer ifade varsa ifadenin değeri döner. Eğer birden fazla ifade varsa tüm ifadelerin değerleri dizi olarak döner.

Lua'da birden fazla değer dönüşü vardır.

return 3, sum, index gibi bir dönüş yapıldığında bunun çağrısı a, b, c = fun() şeklinde olabilir.