

Programlama Dilleri (315)

Dr. Öğr. Üyesi Ahmet Arif AYDIN

Sub Programs

(Alt Programlar)

- Komut seviyeli kontrol yapıları
- Goto Statement considered harmful
- n-way, two way selection
- Multiple selection
- Iterative statements (loop)
- Break, continue
- guarded commands

Programlama dillerine iki temel soyutlama (abstraction) kolaylığı bulunmaktadır

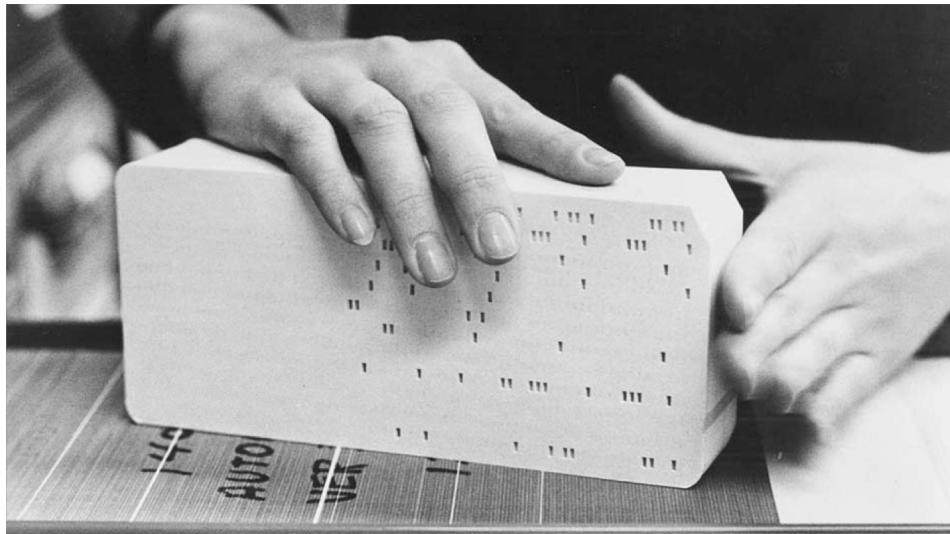
1. İşlem - süreç soyutlama (process abstraction)

- Yüksek seviyeli programlama dilleri ortaya çıktığı ilk zamanlardan beri kullanılmaktadır.
- Altprogramlar yapısıyla işlem soyutlamaktadır.
- Alt program çağrıları da bir abstraction'dır.

2. Veri soyutlama (data abstraction)

- 1980 li yıllarda itibaren önem kazanmaya başlamıştır.

Babbage's Analytical Engine, 1840



punch cards

Günümüzde altprogramlar
yardımıyla kod yazılmakta



5MB
luk kod

Genel Özellikler

- Her alt programın sadece bir başlangıç noktası bulunur.
- Bir altprogram çağrıldığında, çağrıran program alt programın çalışmasının bitinceye kadar askıya alınır (suspend)
- Alt programların çalışması tamamlandığında kontrol çağrıran programa geri döner.

Temel Tanımlamalar

- Altprogram tanımı (*subprogram definition*): Alt programın arayüzüünü ve gerçekleştirdiği işlemleri tanımlar
- Altprogram çağrısı (*subprogram call*): altprogramın çağrılmaya isteği
- Çağrılan ve halen çalışmasını devam eden altprogram aktif altprogram olarak adlandırılır.
- İki çeşit altprogram bulunmaktadır
 - prosedürler ve fonksiyonlar

Alt Programların Temelleri (Fundamentals of Subprograms)

Altprogram başlığı (*subprogram header*): Alt programın tanımının ilk kısmıdır.

Altprogramın adı , çeşidi (prosedür veya fonksiyon) ve parametreleri tanımlar.

PYTHON

```
def altprogram1 (parametreler):  
    ifadeler
```

C

```
void altprogram2 (parametreler)  
{  
    ifadeler  
}
```

Python '*indentation*' , Ruby 'end' ve C tabanlı diller '}' simbolü ile altprogramın alanını tanımlanır

Alt Programların Temelleri (Fundamentals of Subprograms)

```
if şart
    def fun(...):
    ...
    işlemler
    ...
else
    def fun(...):
...

```

```
def anaprogram():
    isim= 'Programlama Dilleri'
    print('isim: ',isim)
    isimgönder(isim)

def isimgönder (parametre):
    print ('gönderilen isim:', parametre)

anaprogram()
```

Alt Programların Temelleri (Fundamentals of Subprograms)

Altprogram tanımında bulunan parametre profili parametrelerin sayısını , sırasını ve tipini tipini ve belirler.

```
def test
    i = 100
    j = 200
    k = 300
    return i, j, k
end

var = test
index=0
var.each do |key|
    puts "#{index} . deger #{key}"
    index+=1
end
```

\$ruby main.rb

0	.	deger	100
1	.	deger	200
2	.	deger	300

Alt Programların Temelleri (Fundamentals of Subprograms)

Altprogram protokolü : altprogramın gönderdiği tip ve parametreleridir.

- Statik tip kontrolü yapan dillerde (C , C++) fonksiyonların ve altprogramların tanımlanması (declarations) gerekmektedir.
- Bu tanımlamalara prototip (prototype) adı verilir.
- Diğer dillerde altprogramların çağrılmadan önce tanımlanma zorunluluğu yoktur.

Alt Programların Temelleri (Fundamentals of Subprograms)

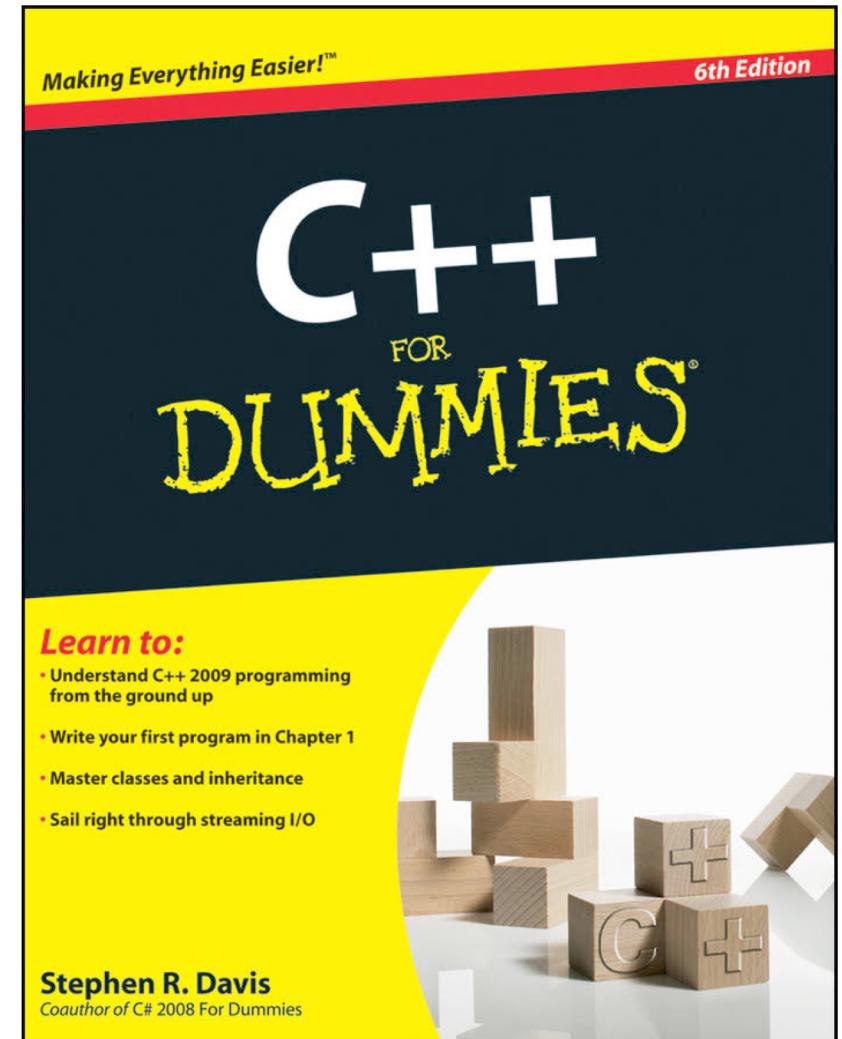
Bir altprogramın işlenmesi gereken veriye erişimi iki yolla olur:

1. Local olmayan değişkenler vasıtasıyla (global)
2. Parametreler yardımıyla

Alt Programların Temelleri (Fundamentals of Subprograms)

Altprogramın tanımında bulunan parametrelere formal parameters denir

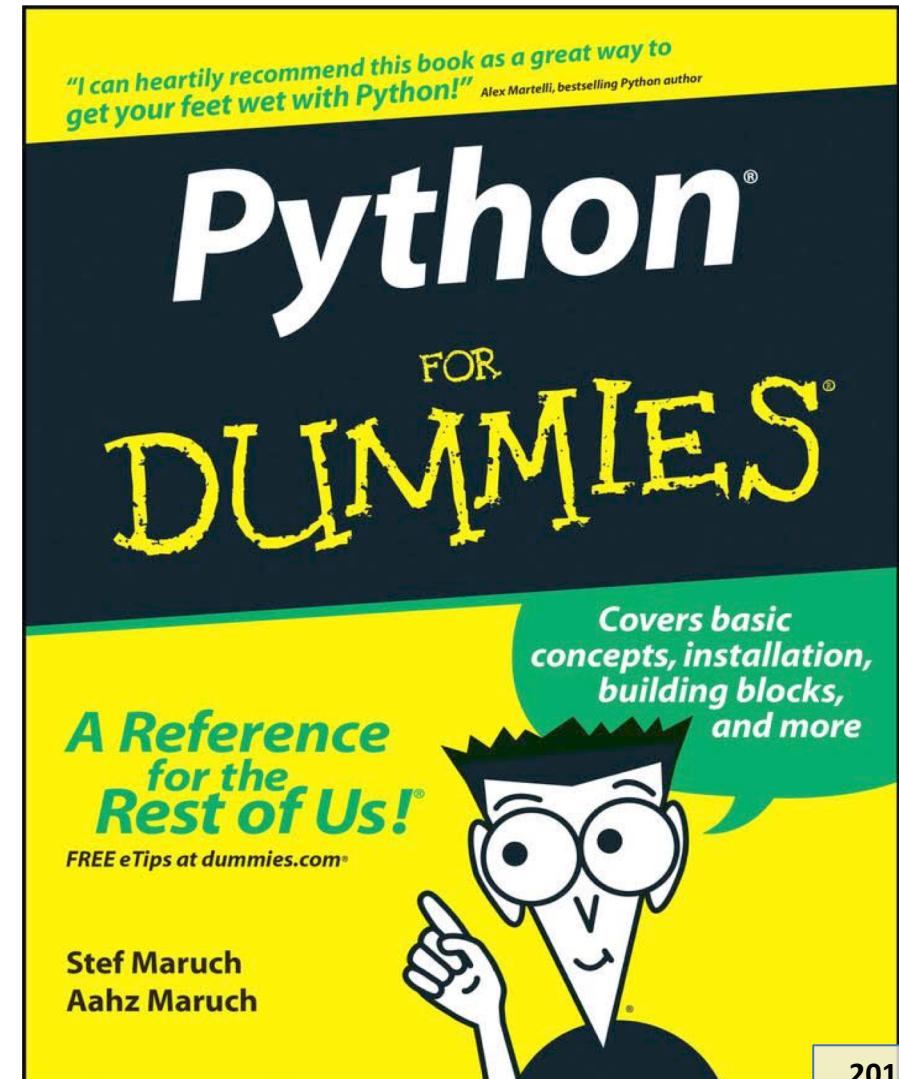
- Dummy variables
- Altprogram çağrılığında kullanılırlar



Alt Programların Temelleri (Fundamentals of Subprograms)

Altprogram çağrıları

- altprogramın ismini ve parametre(ler) i içerir.
- Altprogram çağrısında bulunan parametrelere actual parameters denir.



Alt Programların Temelleri (Fundamentals of Subprograms)

Parametre sayısı arttığında parametre sırası karıştırılabilir. Bu problemin çözümü için anahtar parameter (keyword parameters) ikilileri kullanılır

PYTHON

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

Alt Programların Temelleri (Fundamentals of Subprograms)

Python, Ruby, C++, Fortran 95+ Ada, ve PHP formal parametrelerin varsayılan (default) değerleri bulunabilir.

```
2
3 - def test(d1 = "Python", d2 = "Ruby")
4     puts "1.ifade = #{d1}"
5     puts "2.ifade = #{d2}"
6 end
7 test "Programlama", "JAVA"
8 test|
```

```
$ruby main.rb
1.ifade = Programlama
2.ifade = JAVA
1.ifade = Python
2.ifade = Ruby
```

Alt Programların Temelleri (Fundamentals of Subprograms)

C# altprogramlarında parametre liste olabilir

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

Prosedürler

- Parametre ile gönderilen değerler ile işlem gerçekleştirebilirler
- Geri değer göndermezler

Fonksiyonlar

- Prosedur ile aynı mantıkda çalışmaktadır
- Geriye değer gönderirler

- Bir veya çok parametre gönderme
- Parametre olarak gönderilen değerlerin tip kontrolü
- Local parametreler statik olarak mı dinamik olarak mı değerlendirilecek
- Altprogram tanımı iç içe alabilir mi?
- Altprogram isimleri parametre olarak gönderilebilir mi?
- Çok amaçlı altprogram alabilir mi?

Local Referencing Environments

Yerel değişkenler (local variables)

- Alt programlar içerisinde tanımlanan değişkenler local değişkenlerdir.
- Local değişkenlerin kapsamı alt program ile kısıtlanmıştır.

Statik değişkenler

- Hafızada yer ayrılması işleminin gerektirdiği zaman kaybı yok (avantaj)
- Recursive işlem yapılmıyor.

Local Referencing Environments

Yığıt dinamik (Stack dynamic)

- recursive işlemlerin yapılmasını sağlarlar (avantaj)
- değişken için bellekte yer ayrılması, ilk değer atanması ve bellek yerinin bırakılması maliyetlidir(dezavantaj)
- Butun değişkenlerin yığıt dinamik olması durumunda alt programdaki deger ler unutulabilir

Local Referencing Environments

Yerel değişkenler (local variables)

```
int toplamhesapla(int liste[], int elemansayısı) {  
    static int toplam= 0;  
    int sayıç;  
    for (sayıç = 0; sayıç < elemansayısı; sayıç ++)  
        toplam+= liste [sayıç];  
    return toplam;  
}
```

JAVA methodlarında bulunan local değişkenler statik olmaz sadece yığıt dinamik

Parametre ile değer gönderme (in mode)

- Parametre ile gönderilen değer için hafızada yeni alan oluşturulur
- Oluşturulan parametre alt programda local değişken gibi çalışır.

Hafızada (bellekte) fazla alan kaplayan nesnelerin parametre ile gönderilmesi maliyetlidir (Hafıza-zaman)

Sonucla parametre gönderme (pass-by result - out mode)

- Altprograma herhangi bir değer gönderilmemektedir
- Parametre ile değer göndermenin dezavantajları burda da geçerlidir

```
void strange (out int x, int index)
{
    x = 17;
    index = 42;
}
...
değer = 21;
k. strange (liste[değer], değer);
```

C#

Değer ve sonuc (pass-by-value-result) (in out mode)

- Parametre değerleri ve sonuc kopyalanır.
- Önceki iki yöntemin birleşimidir.
- Parametre ile gönderilen değer alt programda işlendikten sonra tekrar geri gönderilir.

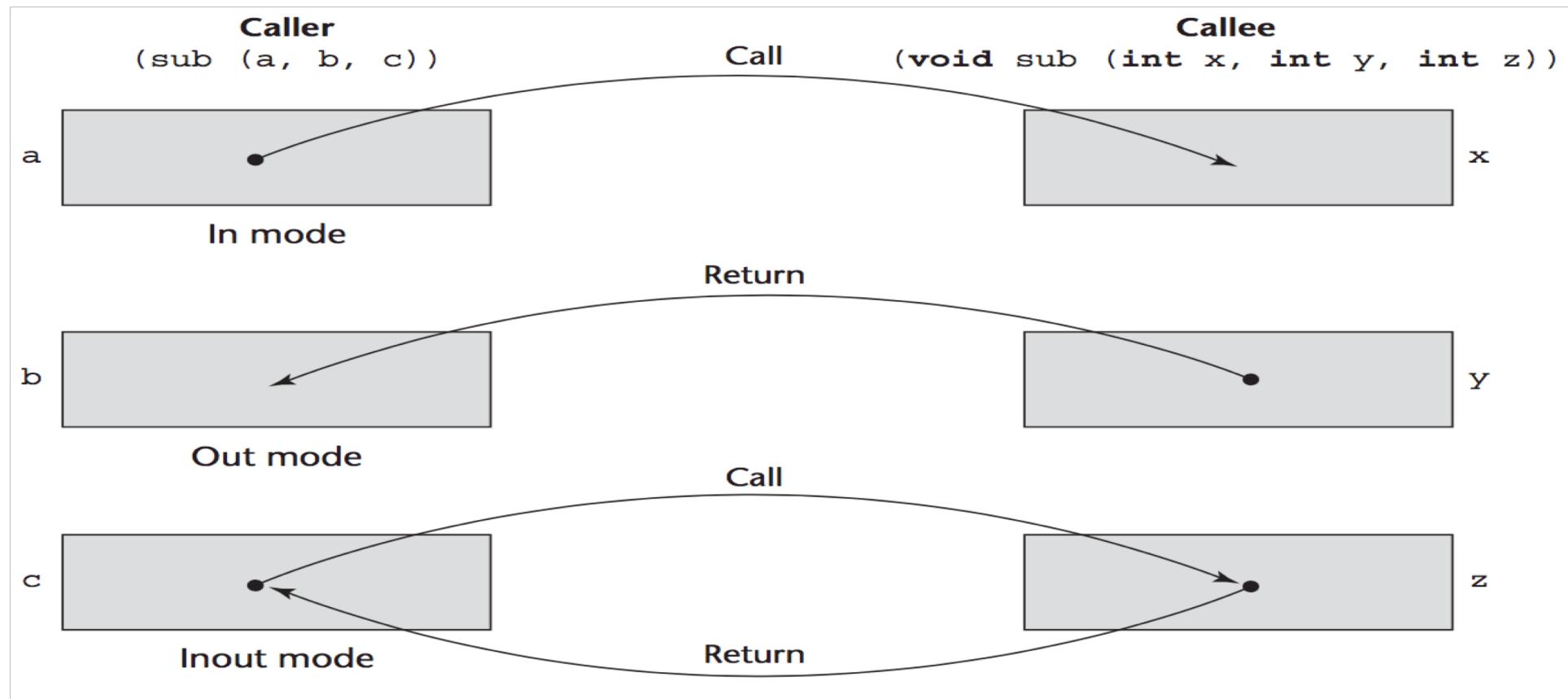
Parametre Gönderme Yöntemleri (Parameter-passing methods)

pass-by-reference (inout-mode)

- Inout modununun diğer bir vesiyonudur.
- Gerçek veriyi tutan belleğe bağlantı yolunu gönderilir
- Alt program ana programın eriştiği gerçek veriye erişir
- Avantaj
 - Veriyi gönderme yerine, veriyi saklayan değişkenin adresi gönderilir
 - Gönderme yöntemi hızlı, zaman ve alan kaybı yok
- Dezavantaj
 - Veriye erişim daha yavaş (dolaylı adreslemeden dolayı)
 - Alias oluşturduğu için programın okunması zorlasır ve alt programın ana programındaki veriyi değiştirmeye imkanı verilir

```
void test(int &first, int &second)  
...  
test(a, b)
```

Parametre Gönderme Yöntemleri (Parameter-passing methods)



Ana programın parametreler ile alt programlarla iletişimini çalışma zamanı yığın (run-time stack) aracılığıyla gerçekleştirir

Parametre Gönderme Yöntemleri (Parameter-passing methods)

Programlama Dili	Yöntem	Örnek
C	Pass-by-value Pass-by-reference (pointer)	<pre>void fun(int p2, int * p3) { ... }</pre> <pre>void değiş(int a, int b){ int aradeger= a; a = b; b = aradeger; }</pre> <pre>void değiş(int *a, int *b){ int aradeger = *a; *a = *b; *b = aradeger; }</pre>

Parametre Gönderme Yöntemleri (Parameter-passing methods)

Programlama Dili	Yöntem	Örnek
C++	Pass-by-value Pass-by-reference (pointer) Referans tipi (alias oluşturuyor) Parametreler sabit olarak tanımlanabilirler	<code>void fun(const int &p1, int p2, int &p3) { ... }</code>

```
void değiş(int &a, int &b) {  
    int aradeğer = a;  
    a = b;  
    b = aradeğer;  
}
```

Parametre Gönderme Yöntemleri (Parameter-passing methods)

Programlama Dili	Yöntem	Örnek
JAVA	Pass-by-value (bütün parametreler)	Alt program parametreleri ana programda değişkenleri direkt olarak değiştiremez.
Python, Ruby	Pass-by-assignment	İşlem atama ile gerçekleşir. Gerçek parametrenin değeri formal parametreye atanır.

Parametrelerde Tip Kontrolu (Parameter type checking)

- Fortran ve ilk ortaya çıkan C de parametre tip kontrolü yok
- C++ da alt programlarda parametrelerin tipi belirtilmek zorundadır.
- Ruby ve Python da parametrelerin tip kontrolü yapılmaz (nesnelerin tipi bulunur değişkenlerin olmaz)
- Perl, JavaScript, ve PHP de parametre tip kontrolü bulunmaz

Alt program olan parametreler

- Alt program isimleri parametre olarak gönderilebilir.
- Matematiksel fonksiyonlar da kullanılır (özel amaçlı)
- C ve C++ da alt program isimleri başka bir alt programa parametre olarak gönderilmesi için pointerlar kullanılır.

Çok Amaçlı (overloaded) Altprogramlar

- Çok amaçlı altprogramların birden fazla anlamı bulunmaktadır. Terimlerin tipine göre yapılacak olan işlem belirlenir ve birden fazla aynı isimde altprogram bulunur.
- Parametrelerin sayısı, sırası, tipi ve dönüş değerine göre farklı versiyonlar bulunmaktadır.
- C++, Java, Ada, ve C# da önceden tanımlanan çok amaçlı altprogramlar bulunmaktadır.

```
public class test {  
    test() {  
        ..  
    }  
    test(String s) {  
        ..  
    }  
    test(int i) {  
        ..  
    }  
    public static void main(String args[]) {  
        test nesne1= new test()  
        ..  
        test nesne2= new test("PL")  
        ..  
        test nesne3= new test(335)  
    }  
}
```

Generic Subprograms

- Reusability (tekrar kullanma) yazılım geliştirmede çok önemlidir.
- Generik (polymorphic) altprogram aynı işlemi farklı tipler üzerinde gerçekleştirilebilirler
- Çok amaçlı altprogramlarda ad hoc polymorphism (özel amaçlı çok biçimlilik) özelliği bulunur.
- Parametric polymorphism ile aynı isimli birden fazla alt program tanımlanabilir ve parametre tipi farklı olarak tanımlanabilir.
- Python ve Ruby de parametrelerin tipi olmadığı için çok amaçlı alt programlar yazılabılır.

Generic Subprograms

C++

```
#include <iostream>
using namespace std;
template <class T>
T GetMax (T a, T b) {
    T result;
    result = (a>b)? a : b;
    return (result);
}
```

..

```
..
int main () {
    int i=5, j=6, k;
    long l=10, m=5, n;
    k=GetMax <int> (i,j);
    n=GetMax<long>(l,m);
    cout << k << endl;
    cout << n << endl;
    return 0;
}
```

Fonksiyonlardaki Tasarım Problemleri

Yan etkiler

- Ortadan kaldırmak için parametreler in-mode da olmalı.
- Pass-by-value , pass by reference alias oluşturduğundan yan etkileri olur.

Geri gönderilen değerlerin tipleri

- C , C++ geri gönderilen değer fonksiyon tanımında belirlenir
- Ada, Python ve Ruby fonksiyonları her tipde değer geri gönderebilir.
- Java ve C# metodları her tipde sonuc gönderebilir.

Kullanıcı Tanımlı Çok Amaçlı operatorler

User defined overloaded operators

Ada, C++, Python, ve Ruby de operatorler çok amaçlı olarak tanımlanabilirler.

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

Closures

Değişken gibi saklanabilen fonksiyona **closure** denir

```
function() {  
    var a = 1;  
    console.log(a); // çalışır  
}  
console.log(a); // hata
```

javascript

```
var a = 1;  
function() {  
    console.log(a); // çalışır  
}  
console.log(a); // çalışır
```

```
outer = function() {  
    var a = 1;  
    var inner = function() {  
        console.log(a);  
    }  
    return inner; // fonksiyonu gönderir  
}  
var fnc = outer(); //  
fnc();
```

Fonksiyonel programlama dilleri ve C# da bulunur

Coroutines

- Alt programların özel bir çeşididir.
- Çağırılan ve çağrılan alt programlar aynı seviyelerdirler.
- Korutin'in çalışma mekanizmasına simetrik ünite control modeli denir.
- Birden fazla giriş noktası olabilir.
- Korutin'in nın çağrılmaması resume olarak bilinir.
- Korutin tekrar çağrıldığında kaldığı yerden devam eder.
- İşletim sisteminin çalışması Korutin'in çalışması biçimdedir.

```
sub co1(){  
    ...  
    resume co2();  
    ...  
    resume co3();  
    ...  
}
```

Coroutines

