315 Programlama Dilleri

Yrd. Doç. Dr. Ahmet Arif AYDIN

Sub Programs

(Alt Programlar)

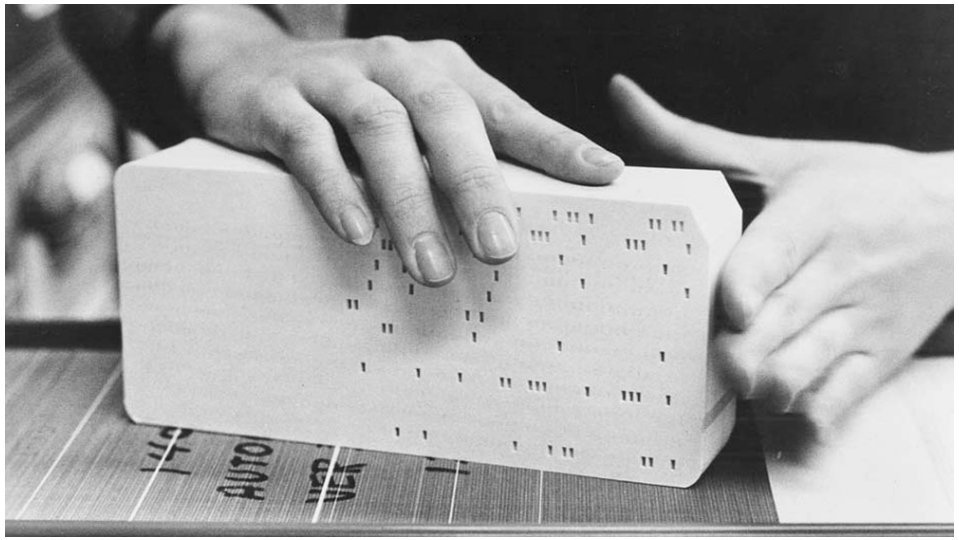
Giriş

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 çağrısı abstractıon dır.
2. Veri soyutlama (data abstraction)
   - 1980 li yıllardan itibaren önem kazanmaya başlamıştır.

Giriş

Babbage’s Analytical Engine, 1840



punch cards

Günümüzde altprogramlar yardımıyla kod yazılmakta

5MB
lık kod

Fundamentals of Subprograms

Altprogramların temelleri

Genel Özellikler

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

Fundamentals of Subprograms

Altprogramların temelleri

Temel Tanımlamalar

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

Fundamentals of Subprograms

Altprogramların temelleri

Temel Tanımlamalar

- 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
}
```

Fundamentals of Subprograms

Altprogramların temelleri

Temel Tanımlamalar

- 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 *'identation'* , Ruby 'end' ve C tabanlı diller ' }' sembolü ile altprogramın alanını tanımlanır

Fundamentals of Subprograms

Altprogramların temelleri

Temel Tanımlamalar

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

```
def main():
      isim= "Programlama Dilleri"
      print "isim: ",isim
      isimgönder (isim)

def isimgönder (n):
      print "gönderilen isim: ", n

main()
```

Fundamentals of Subprograms

Altprogramların temelleri

Temel Tanımlamalar

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

Fundamentals of Subprograms

Altprogramların temelleri

Temel Tanımlamalar

```ruby
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
```

Fundamentals of Subprograms

Altprogramların temelleri

Temel Tanımlamalar

Altprogram protokolü :

altprogramın gönderdiği tip ve parametreleridir.

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

Fundamentals of Subprograms

Altprogramların temelleri

Parametreler

Altprogramın işlenmesi gereken veriye erişimi iki yolla olur

1. Local olmayan değişkenler vasıtasıyla
2. Parametreler yardımıyla

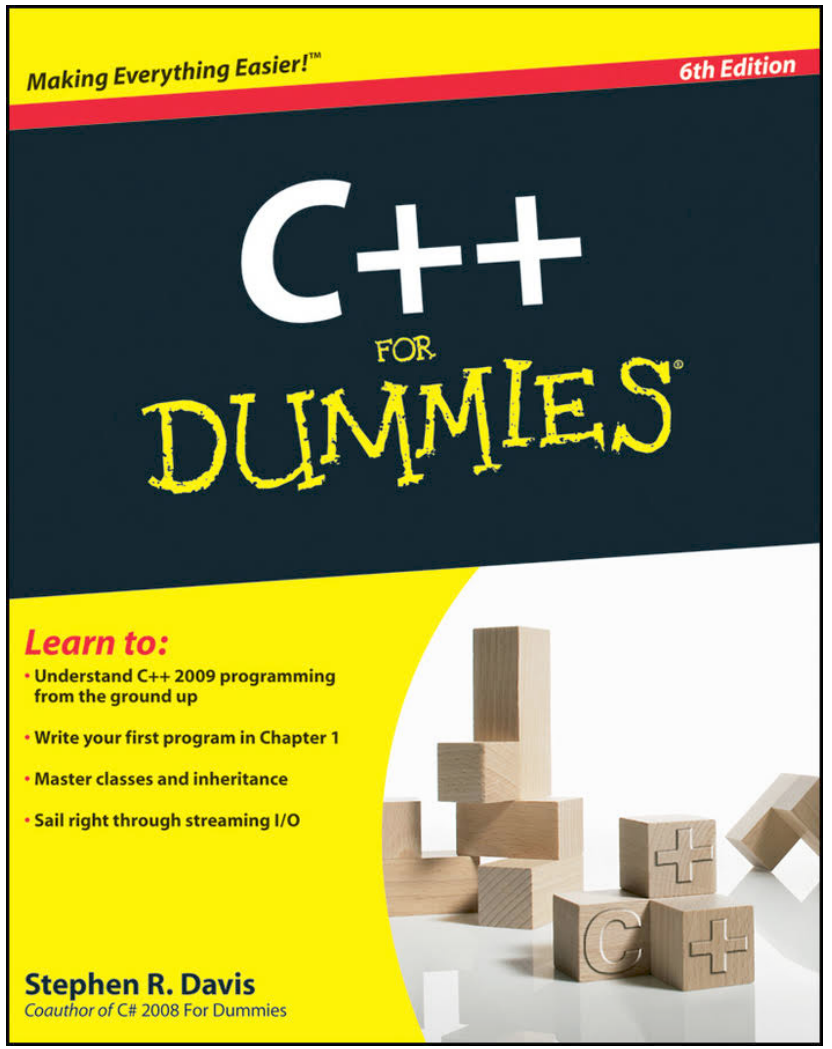
Fundamentals of Subprograms

Altprogramların temelleri

Parametreler

Altprogramın tanımında bulunan parametrelere formal parameters denir

- Dummy variables
- Altprogram çağrıldığında kullanılırlar



Fundamentals of Subprograms

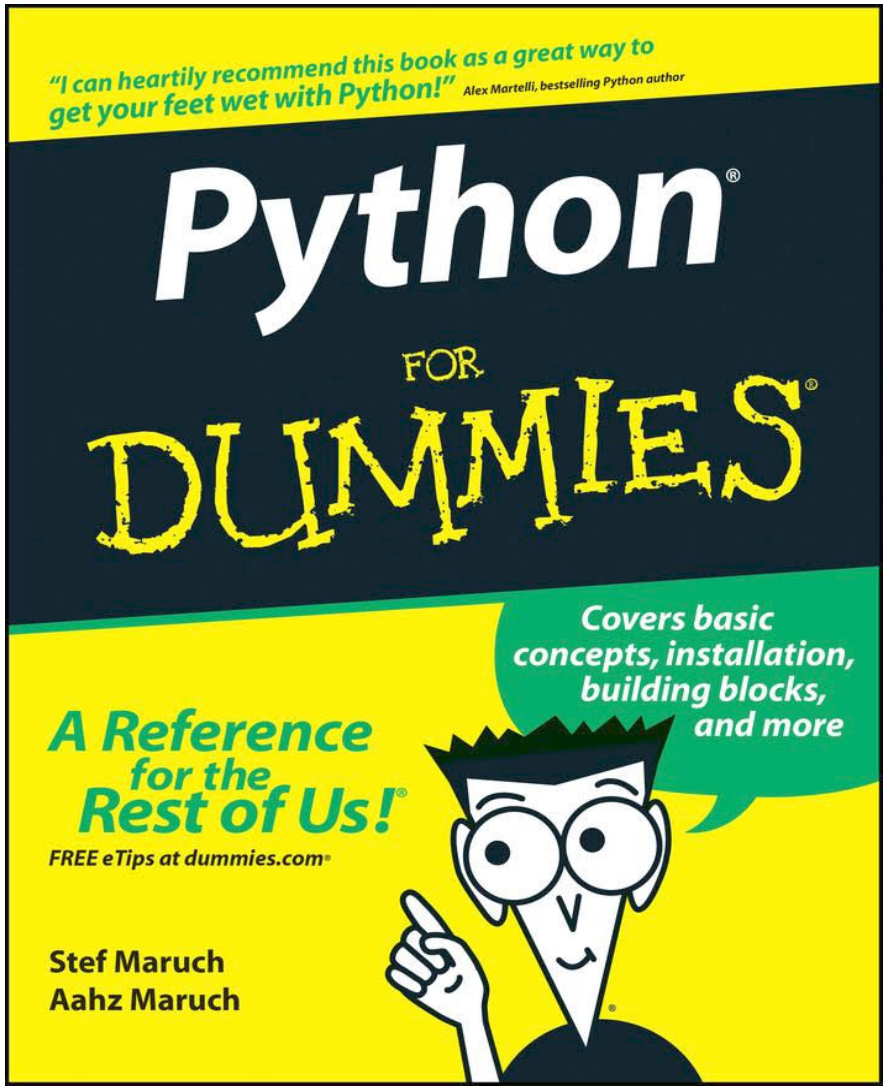
Altprogramların temelleri

Parametreler

Altprogramın tanımında bulunan parametrelere formal parameters denir

- Dummy variables
- Altprogram çağrıldığında kullanılırlar

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



Fundamentals of Subprograms

Altprogramların temelleri

Keyword parameters

- Parametre sayısı arttığında parametre sırası karıştırılabilir.
- Bu problemin çözümü için
  - Keyword parameters

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

Fundamentals of Subprograms

Altprogramların temelleri

Keyword parameters

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

```ruby
def test(d1 = "Python", d2 = "Ruby")
    puts "1.ifade = #{d1}"
    puts "2.ifade = #{d2}"
end
test "Programlama", "JAVA"
test
```

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

Fundamentals of Subprograms

Altprogramların temelleri

C# altprogramlarında parametre liste olabilir

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

Fundamentals of Subprograms

Altprogramların temelleri

Prosedürler

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

Fonksiyonlar

- Prosedur ile aynı mantıkda calımaktadır
- Geriye değer gönderirler

Design İssues of Subprograms

Altprogramların tasarım problemleri

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

Local Referencing Environments

Altprogramların tasarım problemleri

Yerel değişkenker (local variables)

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

Statik

- hafızada Yer ayrılması ıslemının gerektırdıgı zaman kaybı yok. (avantaj)
- Recursive işlem yapılmıyor.

Yıgıt dinamik (Stack dynamıc)

- recursive işlemlerin yapılmasını sağlarlar (avantaj)
- değişken için bellekte yer ayrılması, ilk değer atanması ve bellek yerinin bırakılması maliyetlidir(dezavantaj)
- Butun değişkenlerın yıgıt dınamık olması durumunda alt programdaki deger ler unutulabılır,

Local Referencing Environments

Altprogramların tasarım problemleri

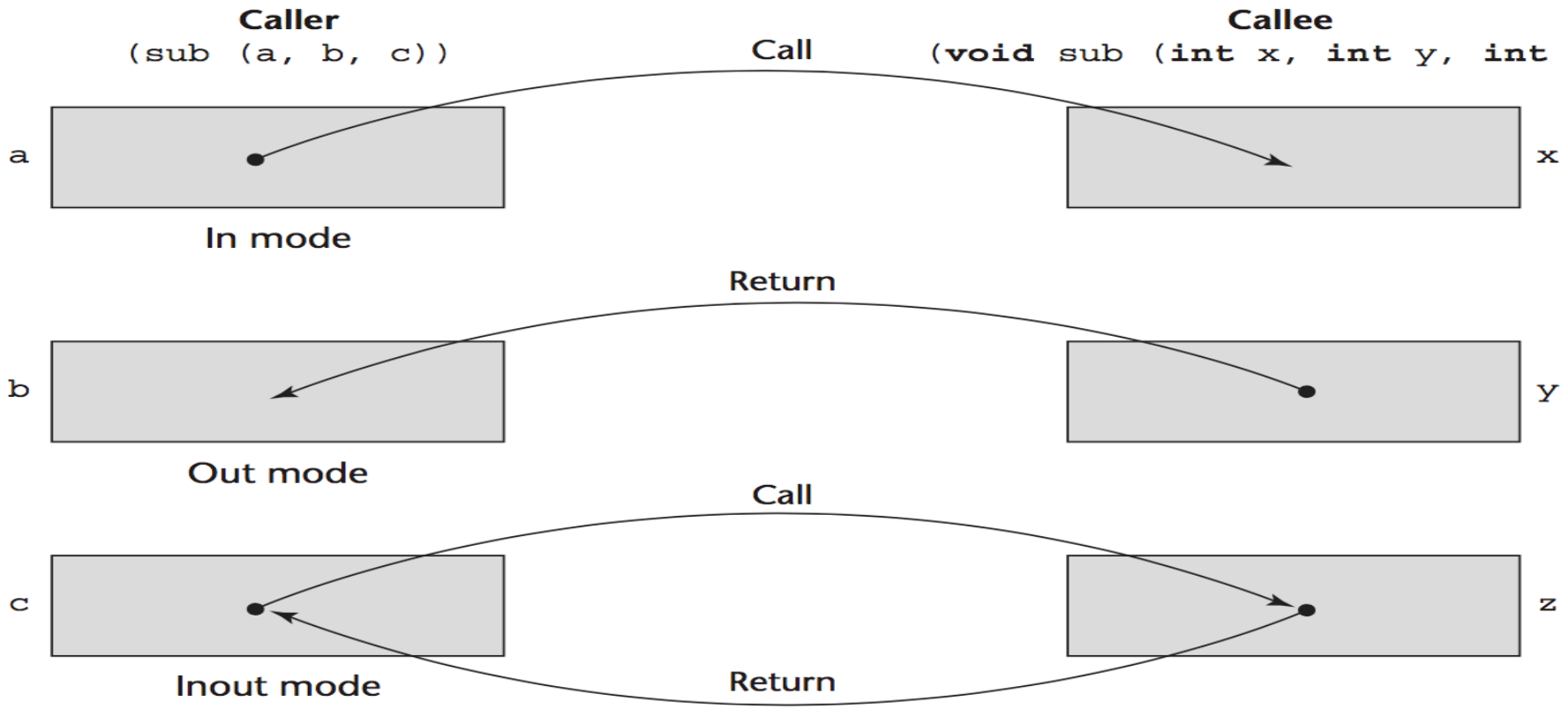
Yerel değişkenker (local variables)

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

JAVA methodlarında bulunan local degişkenler statik olmaz sadece yıgıt dinamik

Paramater-passing methods

Parametre gönderme yöntemleri



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

Paramater-passing methods

Parametre gönderme yöntemleri

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)

Paramater-passing methods

Parametre gönderme yöntemleri

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

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

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

C#

Paramater-passing methods

Parametre gönderme yöntemleri

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

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

Paramater-passing methods

Parametre gönderme yöntemleri

pass-by-reference (inout-mode)

- Inout modununun diğer bir vesiyonudur.
- Gercek veriyi tutan belleğe bağlantı yolu gönderilir
- Alt program ana programın eriştiği gercek veriye erişir

Paramater-passing methods

Parametre gönderme yöntemleri

pass-by-reference (inout-mode)

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

- Inout modununun diğer bir vesiyonudur.
- Gercek veriyi tutan belleğe bağlantı yolu gönderilir
- Alt program ana programın eriştiği gercek veriye erişir
- Avantaj
  - Veriyi gonderme yerine, veriyi saklayan değişkenin adresi gönderilir
  - Gönderme yöntemi hızlı , zaman ve alan kaybı yok
- Dezavantaj
  - Veriye erişim daha yavaş (dolaylı adreslemeden dolayı)
  - Alias aluşturdugu için programın okunması zorlasır ve alt programın ana programdaki veriyi değiştirme imkanı verilir

Paramater-passing methods

Parametre gönderme yöntemleri

Programlama Dili	Yöntem	Örnek
C	Pass-by-value Pass-by-reference (pointer)	**void** fun(**int** p2, **int** * p3) { ... }

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

```
void değiş(int *a, int *b) {
    int aradeger = *a;
    *a = *b;
    *b = aradeger;
}
```

Paramater-passing methods

Parametre gönderme yöntemleri

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

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

Paramater-passing methods

Parametre gönderme yöntemleri

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.

Paramater type checking

Parametrelerde tip kontrolü

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

Parameters that are subprograms

- 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.

Overloaded Subprograms

- Çok amaçlı operatorlerin birden fazla anlamı bulunmaktadır. Terimlerin tipine göre yapılacak olan işlem belirlenir.
- 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 gercekleştirebilirler
- Çok amaçlı altprogramlar ad hoc polymorphism (özel amaçlı cok 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 olmadıgı için cok amaclı alt programlar yazılabilir.

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;
}
```

http://www.cpluspl
us.com/doc/oldtut
orial/templates/

Design issues for functions

Fonksiyonlardaki tasarım problemleri

Yan etkiler

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

Geri gönderilen degerlerin 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.

Geri gonderilen degerlerin sayısı: 1 veya daha fazla

User defined overloaded operators

Kullanıcı tanımlı çok amaçlı operatörler

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

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

Closures

Değişken gibi saklanabilen fonksiyona denir

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

```
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

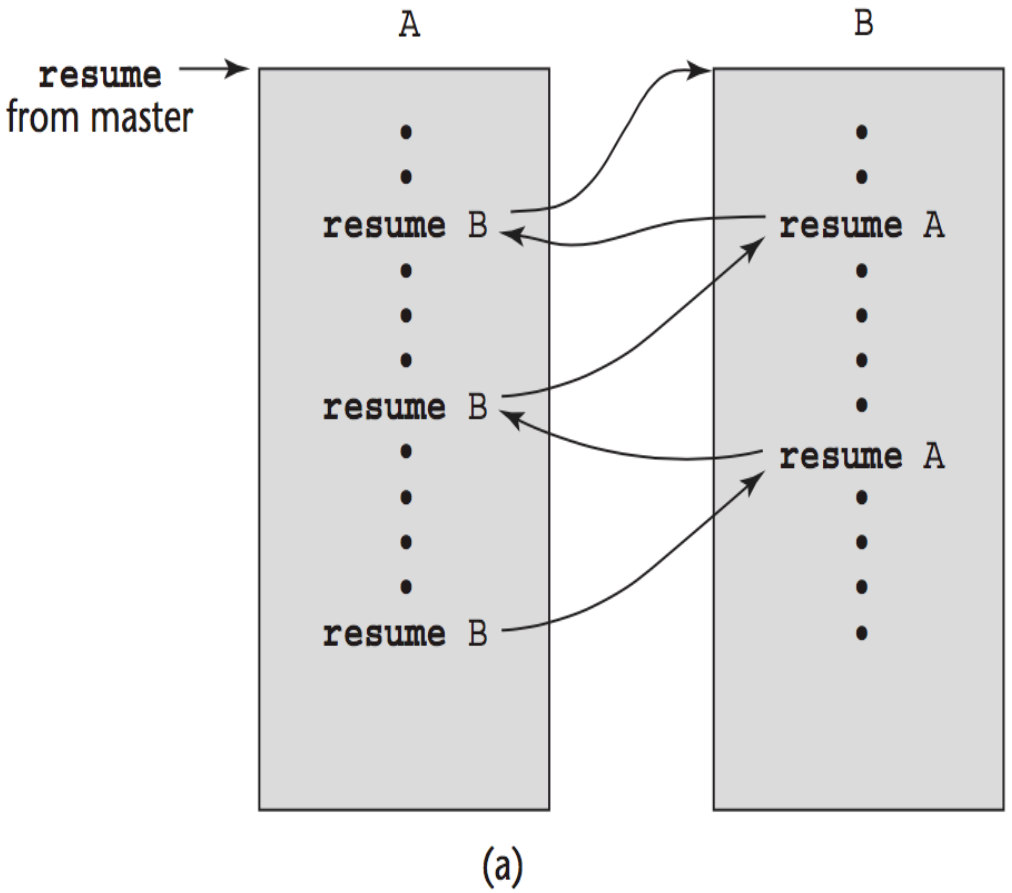
https://stackoverflow.com/questions/36636/what-is-a-closure

Coroutines

- Alt programların özel bir çeşididir.
- Çağıran ve çağrılan alt programlar aynı seviyededirler.
- Korutin'in çalışma mekanizmazına simetrik ünite control modeli denir.
- Birden fazla giriş noktası olabilir.
- Korutin'in nın çağrılması resume olarak bilinir.
- Korutin tekrar cağrıldıgında kaldıgı yerden devam eder.
- İşletim sistemının çalışması Korutin'in çalışması biçimdedir.

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

Coroutines



(a)