BLG339
PROGRAMLAMA DİLİ KAVRAMLARI

Hafta 1

Yrd. Doç. Dr. Melike Şah Direkoğlu

Alındığı kaynak:
Addison-Wesley's Programming Language Concepts slaytları ve Prof. Dr. Tuğrul Yılmaz' ın ders notlarından faydalanarak hazırlanmıştır.

Konular

- Programlama Dilleri Kavramlarının Çalışılma Nedenleri
- Programlama Alanları
- Dil Değelendirme Kriterleri
- Dil Tasarımındaki Etkileri
- Dil Kategorileri ve Dillerin Tarihçesi
- Dil Tasarımındaki Getiri-Götürü İlişkileri
- Gerçekleştirim Metotları
- Programlama Çevreleri

Neden Programlama Dilli Kavramları Dersi?

- Fikirlerimizi uygularken daha kolay ve daha iyi yapabilmek için.
  - Programalama dillerinin detaylarını bilerek yazılım zenginleştirilebilir.
- Seçeneklerimizin ne olduğunu bilirsek iyiyi seçebiliriz.
  - Bilgimizi artırarak eldeki probleme en uygun programalama dilini seçebiliriz.
- Dil öğrenmede yetkinlik. Dillerin özelliklerini bilmeyen, belli bir dille çalışmaya alışmış kişi, farklı bir dili öğrenmesi gerektiğinde zorlanır.
  - Örnek: Nesneye yonelik programlama kavramını bilen bir kişi, Java'yı bu konsepti bilmeyen bir kişiye gore daha kolay öğrenebilir.

Neden Programlama Dilli Kavramları Dersi? (Devam)

- Belli bir dilin önemli özelliklerini anlayarak daha iyi kullanabilmek için.
  - Diller komplex yapılardan olusur. Fakat önemli özellikler etkin kullanılarak yazılım gelistirilebilir.
- Dilleri daha iyi değerlendirebilirsek, doğru seçimler yaparız, doğru teknolojilerin gelişmesine destek olmuş oluruz.
- Gerçekleştirimin anlaşılmasıyla programlama dilini daha iyi anlama.
  - Örnek: Subprogramlar sıklıkla çağrılırsa, program hızı duşer. Bunu bilirsek daha iyi program tasarımı yapabiliriz.
- Hata bulurken özelliklerini bilmemiz faydalıdır.
- Özellikleri öğreniriz, olmayan özelliklerine öykünürüz (emulate).

Programlama Alanları

- Bilimsel uygulamalar (scientific applications)
  - Büyük sayıda noktalı hesaplama yapma
  - Doğru hesaplama en önemli özellik
  - Fortran (1950 ler), Algol 60 (1960 lar)
- İş Uygulamaları (business applications)
  - Rapor oluşturma, ondalık sayı ve karakterlerin kullanımı
  - COBOL (1960 lar) halen en populeri
- Yapay Zeka
  - Sayılar yerine semboller kullanılır, dizi yerine bağlantılı bilgi
  - Programlar çok daha esnek yapıya sahip olmalıdır; program çalışırken yeni kod uretip çalıştırabilmelidir.
  - LISP (1965), Prolog (1970 ler)

Programlama Alanları (Devam)

- Sistem Programlama
  - Sürekli kullanım nedeniyle hızlı ve verimli çalışma gereksinimi
  - IBM'in ilk sistem programi PL/I (1970 ler), C (1970ler)
  - Hemen hemen tum işletim sistemleri C veya C++ ile yazılmıştır.
  - UNIX tamamen C ile yazılmıştır.
- Web Yazılımı
  - Çeşitli diller:
    - markup (örn. HTML, XHTML) bir programlama dili degildir
    - scripting languages (örn., PHP, Javascript) dinamik icerik icin HTML dokuman ı na program kodları eklemek için
    - genel-amaçlı (örn. Java (applets, servlets))

Dil Değelendirme Kriterleri

- Okunabilirlik: Programlar kolay okunabilir ve anlaşılır olmalı
- Yazılabilirlik: Program oluşturmak için yazımının kolay olması
- Güvenilebilirlik: teknik şartnamelere uygunluğu; tanımlara uyması
- Maliyet: son toplam maliyet

Değerlendirme Kriteri: Okunabilirlik

- Bütünün basitliği
  - Yönetilebilir özellikler ve yapılar
  - Aynı işi yapan özelliklerin az olması
    - Örnek: c = c + 1; c+ = 1; c++; ++c;
  - Minumum operatörlerin aşırı yüklenmesi (+ (toplam) integer, double ve dizilerin toplanması için kullanılır.)
- Orthogonality
  - Ilkel yapıların küçük sayıdaki yollar ile bir araya getirilerek birleştirilebilmesi (örn: 3 sayı ve 1 karakter veri tipi vede dizi ve göstergeleri kullanarak birçok kontrol ifadesi yazılabilir.)
  - Birbirinden bağımsız yapıların varlığı ve tanımlanması
- Kontrol ifadeleri
- – İyi bilinen kontrol ifadelerinin varlığı(örn., while ifadesi)
- Veri Tipleri ve yapıları
  - Veri yapılarının tanımlamak için yeterli sayıda kolaylığın olması
- Söz dizim tasarımı
  - Bileşik ifadeleri oluşturmak için özel kelime ve metotların olması (class, for, while)
  - Biçim ve anlam: kendi-kendini tanıtan yapılar, anlamlı anahtar kelimeler (static)

Değerlendirme Kriteri: Yazılabilirlik

- Basitlik
  - Az yapıcının olması, küçük sayıda ilkellerin olması, bunları birleştirecek kuralların az olması
- Soyutlama desteği
  - Detayları yok sayarak karmaşık yapı ve işlemlerin tanımlanmasına ve kullanma yeteneği
- Anlamlılık
  - İşlemleri tanımlamak için uygun yolların olması
  - örneğin: for ifadesinin birçok modern dile katılması
- Okunabilirlik ve Yazılabilirlik
  - Bir algoritmayı doğal bir şekilde ifade yolları bulunmayan diller ister istemez doğal olmayan yaklaşımları kullanacaktır, böylece de okunabilirlik azalacaktır.

Değerlendirme Kriteri: Güvenilirlik

- Tip kontrolü
  - Tip hataları için test etme
- İstisna(Exception) işleme
  - Çalışma zamanı hatalarının kesilmesi ve düzeltici önlemlerin alınması
- Örtüşme
  - Aynı bellek bölgesini işaret eden iki yada daha fazla farklı referansın olabilmesi iyi değildir.

Değerlendirme Kriteri: Maliyet

- Dili kullanmak için programcıların eğitimi
- Program yazma (özel uygulamalara kapalılık)
- Programları derleme
- Programları yürütme
- Dil gerçekleştirme sistemi: derleyicilerin bulunması ve kullanılabilirliği
- Güvenilirlik: zayıf güvenilirlik yüksek maliyetlere neden olur
- Programların bakımı

Değerlendirme Kriteri: Diğerleri

- Taşınabilirlik
  - Bir programın bir gerçekleştirimden başka bir gerçekleştirime kolaylıkla taşınabilir olması
- Genellik
  - Geniş sahadaki uygulamalara uygulanabilirlik
- İyi tanımlanabilirlik
  - Dilin resmi tanımının tam ve kesin olması

Dil Tasarımının Getiri-Götürüsü

- Güvenilirliğe karşı çalıştırma maliyeti
  - Örneğin: Java dizi içindeki elemanların tamamına ulaşımda referansların ve indislerin kontrol edilmesini talep eder, bu da çalıştırma maliyetini arttırır.
- Okunabilirliğe karşı yazılabilirlik
  - Örneğin: APL birçok güçlü operatör yardımıyla oldukça karmaşık hesaplamaların yapılabilmesine imkan verir fakat okunabilirlik azalır.
- Yazılabilirliğe (esneklik)karşı güvenilirlik
  - Örneğin: C++ işaretçileri güçlüdür ve oldukça esnektir fakat kullanımı güvenilir değildir.

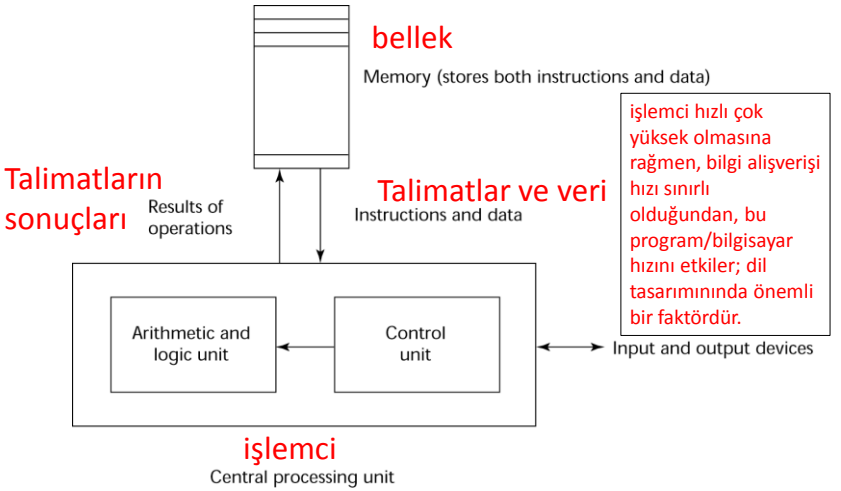
Dil Tasarımını Etkileyenler

- Bilgisayar Mimarisi
  - Diller von Neumann mimarisi olarak bilinen bir bilgisayar mimarisi temelinde geliştirilirler.
- Programlama Metodojileri
  - Yeni yazılım geliştirme metedojileri(örn, nesneye yönelik yazılım geliştirme) yeni programlama paradigmaları ve eklentilerine yol açmıştır

Bilgisayar Mimarisi Etkisi

- İyi bilinen bilgisayar mimarisi: Von Neumann
- Emirsel(Imperative) diller, von Neumann bilgisayarlarının kullanılması nedeniyle yaygın kullanılırlar.
  - Veri ve programlar bellekte saklanır
  - Bellek işlemciden ayrıdır
  - Talimatlar ve veri bellekten işlemciye gider
  - Emirsel diller için temeller
    - Değişkenler bellek hücrelerini modeller
    - Atama ifadeleri veri getirme işini modeller
    - İterasyon etkilidir

The von Neumann Mimarisi



Programlama Metodolojileri Etkisi

- 1950 ve 1960 ların başlarında : Basit uygulamalar vardır ve verimli değildir.
- 1960 ların sonlarında: İnsanların verimlilik ve etkinliği önemli oldu; okunabilirlik ve daha iyi kontrol yapıları oluştu
  - yapısal programlama
  - yukarıdan-aşağı tasarım
- 1970 lerin sonlarında: Süreç yönelimliden veri yönelimli hale gelmişlerdir
  - veri soyutlama
- 1980 lerin ortalarında: Nesne yönelimli programlama
  - Veri soyutlama+ kalıtım+ polymorphism

Dil Kategorileri

- Emirsel/Zorunlu/Buyurgan(imperative)
  - Merkezi özellikleri değişkenler, atama ifadeleri ve döngülerdir
  - Örnek: C, Pascal
- Fonksiyonal(Functional)
  - Hesaplama yapmanın temelinde veriler ve parametrele fonksiyonları uygulamak
  - Örnek: LISP, Scheme
- Mantık(Logic)
  - Kural tabanlı (kurallar belirli sıralama olmadan verilir)
  - Örnek: Prolog
- Nesneye yönelik (Object-oriented)
  - Veri soyutlama, kalıtım, polymorphism
  - Örnek: Java, C++
- İşaretleme (Markup)
  - Yeni; tam bir programlama dili değillerdir fakat web dökümanlarındaki bilginin yerleşimini belirtmede kullanılır.
  - Örnek: XHTML, XML

Programlama Dilleri Taksonomisi

- Emirsel/Zorunlu/buyurgan (imperative) (akış kontrolüne odaklı)
  - Nesneye yönelik (Object Oriented)
- Bildirimci (declarative)
  - Fonksiyonel
  - Veri akışı
  - Mantıksal

Emirsel/Zorunlu/Buyurgan (İmperative) Diller

- Akış kontrolüne odaklı komutlar
  - Yordamsal/yöntemsel (Procedural)
  - Veri üzerindeki eylemi belirler.
  - Assembly, Fortran, Basic, Pascal, C, Bourne Shell
- Nesneye yönelik (Object Oriented)
  - Veriyi gruplandırmaya ve işlemeye dil desteği (Kılıflama (Encapsulation))
  - C++, Java

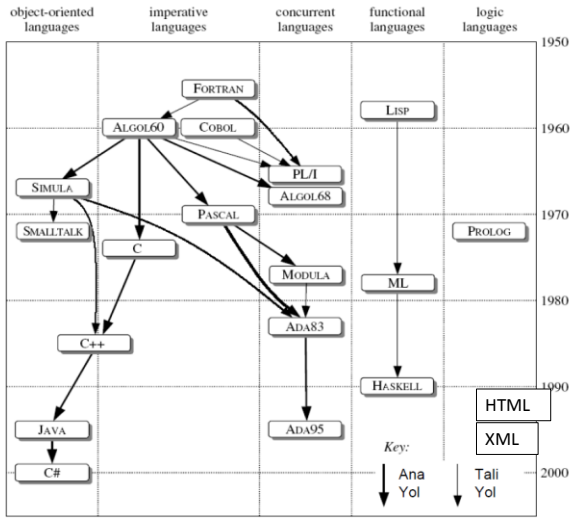
Bildirimci (Declarative) Diller

- Bildirimci = veri güdümlü (Data Driven) Diller
  - Fonksiyonel: Lisp, ML, Haskell
  - Mantıksal, kısıt tabanlı (Logical, Constraint Based)
    - Kuralları koy, başlangıç koşullarını belirle, sonuç için komutları belirle.
      - Prolog and Spread Sheets (Excel)
- İlişkisel (relational) – Veri tabanı sorgulaması (Database Query) - SQL

Programlama Dillerinin Tarihçesi

Dillerin tarihçesi

- **Object-oriented:** Nesne tabanlı.
- **İmperative:** Buyurgan dil. İşlevsel dillerden farklı olarak değişkenlere değerler atayan dil.
- **Concurrent:** Koşut zamanlı diller, koşut zamanlı görev/komut dizisi ile belirginleşir.
- **Functional:** fonksiyonel dil. Lisp, Postscript, Haskell örneklerinde olduğu gibi, bir veri işleme işinde yapılacak işleri salt fonksiyon çağrıları ile ifade eden programlama dili.
- **Logic:** mantıksal dil.



IBM 704 ve FORTRAN (FORmula TRANSlation)

- FORTRAN I – 1957
  - Yeni IBM 704 için gerçekleştirildi
  - isimler 6 karaktere kadardı
  - DO loop
  - Formatted i/o
  - alt programlar
  - arithmetic IF: if(aritmetik ifade) N1,N2,N3
  - veri tipi yok
  - 400 satırdan uzun program nadiren derlendi. Bunun nedeni 704'ün güvenilmezliğiydi.
  - kod hızlıydı.
  - hızla kullanılmaya başladı.
- .....
- Fortran 95 – 1995
- Fortran 2003
- FORTRAN değerlendirme
  - Çok büyük ölçüde değişti ve hala kullanılıyor!

Fortran Kodu

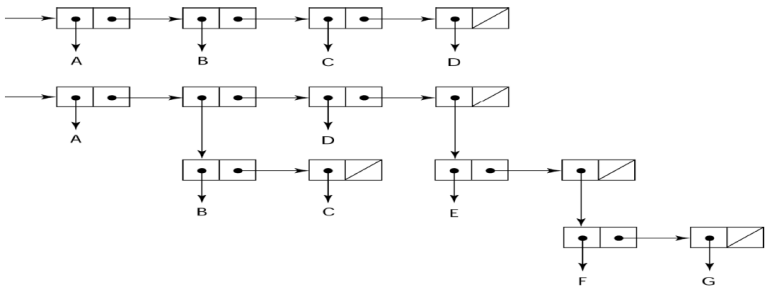
```
! Fortran 95 Example program
!  Input:  An integer, List_Len, where List_Len is less
!          than 100, followed by List_Len-Integer values
!  Output: The number of input values that are greater
!          than the average of all input values
Implicit none
Integer :: Int_List(99)
Integer :: List_Len, Counter, Sum, Average, Result
Result= 0
Sum = 0
Read *, List_Len
If ((List_Len > 0) .AND. (List_Len < 100)) Then
! Read input data into an array and compute its sum
   Do Counter = 1, List_Len
      Read *, Int_List(Counter)
      Sum = Sum + Int_List(Counter)
   End Do
! Compute the average
   Average = Sum / List_Len
! Count the values that are greater than the average
   Do Counter = 1, List_Len
      If (Int_List(Counter) > Average) Then
         Result = Result + 1
      End If
   End Do
! Print the result
   Print *, 'Number of values > Average is:', Result
Else
   Print *, 'Error - list length value is not legal'
End If
End Program Example
```

LISP – 1959

- LISt Processing language (Designed at MIT by McCarthy)
  - İki veri tipi var: atom ve list
  - Sözdizim lambda calculus'a dayanır
  - Fonksiyonel programlamada öncü
  - Değişkenlere gerek yok.
  - Özyineleme (recursion) ve koşullu ifadeler ile kontrol.
  - Yapay zeka için hala dominant.
  - Common LISP, Standard Lisp ve Scheme çağdaş lehçeleri.
  - ML, Miranda, ve Haskell ilgili diller.

LISP – 1959 (Devam)

list (A B C D) ve (A (B C) D (E (F G)))'nin gösterimi



ALGOL 58 ve 60

- ACM and GAMM 4 günlük toplantıda kararlaştırıldı.
- *ALGOL 58 özellikleri*:
  - type kavramı
  - isim boyu serbest
  - Array indeksleri serbest
  - Compound statements (begin ... end)
  - noktalı virgül komut ayıracı
  - atama operatörü: =
  - **if else-if**

ALGOL 58 ve 60 (Devam)

- Algol 60 Pariste bir toplantıda 6 günde geliştirildi.
- *Yeni özellikler*:
  - Blok yapısı (local scope)
  - İki tip parametre geçirme yöntemi
  - Altprogram özyineleme (Subprogram recursion)
- Başarıları:
  - Algoritmaları açıklamak için kullanılması > 20 yıl
  - !!!!Sonraki bütün buyurgan diller takip etti!!!!
  - !!!!İlk makineden bağımsız dil!!!!
  - !!!!Sözdizimi (syntax) resmen tanımlanan ilk dil (BNF)!!!!
  - Bir komite tarafından tasarlanan ilk dil
- - Başarısızlıkları:
  - Geniş olarak kullanılamadı, özellikle ABD'de.
  - *Nedenleri:*
    - i/o yetersizde, karakter seti programların taşınabilirliğini azaltıyordu
    - Çok esnekti, gerçekleştirimi zordu
    - IBM desteğinin olmaması

Algol 60 Kodu

```
comment ALGOL 60 Example Program
 Input:  An integer, listlen, where listlen is less than
         100, followed by listlen-integer values
 Output: The number of input values that are greater than
         the average of all the input values  ;
begin
  integer array intlist [1:99];
  integer listlen, counter, sum, average, result;
  sum := 0;
  result := 0;
  readint (listlen);
  if (listlen > 0) ∧ (listlen < 100) then
    begin
comment Read input into an array and compute the average;
    for counter := 1 step 1 until listlen do
      begin
      readint (intlist[counter]);
      sum := sum + intlist[counter]
      end;
comment Compute the average;
    average := sum / listlen;
comment Count the input values that are > average;
    for counter := 1 step 1 until listlen do
      if intlist[counter] > average
        then result := result + 1;
comment Print result;
    printstring("The number of values > average is:");
    printint (result)
    end
  else
    printstring ("Error-input list length is not legal");
end
```

ALGOL Soyundan Gelen Önemli Programlama Dilleri

- Pascal – 1971 - Wirth
  - Yapısal programlama öğretmek için tasarlandı.
  - Küçük, basit, gerçek anlamda yeni birşey yok.
  - 1970'lerin ortalarından 1990'ların sonuna kadar dil eğitiminde en çok kullanılan dildi.
- C – 1972 - Dennis Ritchie
  - Sistem programlama için tasarlandı
  - Güçlü işleçler fakat zayıf tip kontrolü.
  - Başlangıçta UNIX kanalıyla dağıtıldı.
- Perl – 1987 – Larry Wall
  - Betik (scripting) dili olarak da tanımlanır.
  - Genel ve web amaçlı programlama dili olarak geniş bir şekilde kullanılır.

Pascal Kodu

```
{Pascal Example Program
 Input:  An integer, listlen, where listlen is less than
         100, followed by listlen-integer values
 Output: The number of input values that are greater than
         the average of all input values }
program pasex (input, output);
  type intlisttype = array [1..99] of integer;
  var
    intlist : intlisttype;
    listlen, counter, sum, average, result : integer;
  begin
  result := 0;
  sum := 0;
  readln (listlen);
  if ((listlen > 0) and (listlen < 100)) then
    begin
{ Read input into an array and compute the sum }
    for counter := 1 to listlen do
      begin
      readln (intlist[counter]);
      sum := sum + intlist[counter]
      end;
{ Compute the average }
    average := sum / listlen;
{ Count the number of input values that are > average }
    for counter := 1 to listlen do
      if (intlist[counter] > average) then
        result := result + 1;
{ Print the result }
    writeln ('The number of values > average is:',
             result)
    end { of the then clause of if (( listlen > 0 ... }
  else
    writeln ('Error-input list length is not legal')
end.
```

C Kodu

```
/* C Example Program
 Input:  An integer, listlen, where listlen is less than
         100, followed by listlen-integer values
 Output: The number of input values that are greater than
         the average of all input values */
void main (){
  int intlist[98], listlen, counter, sum, average, result;
  sum = 0;
  result = 0;
  scanf("%d", &listlen);
  if ((listlen > 0) && (listlen < 100)) {
/* Read input into an array and compute the sum */
    for (counter = 0; counter < listlen; counter++) {
      scanf("%d", &intlist[counter]);
      sum += intlist[counter];
     }
/* Compute the average */
    average = sum / listlen;
/* Count the input values that are > average */
    for (counter = 0; counter < listlen; counter++)
      if (intlist[counter] > average) result++;
/* Print result */
    printf("Number of values > average is:%d\n", result);
   }
  else
    printf("Error—input list length is not legal\n");
}
```

Perl Kodu

```
# Perl Example Program
# Input:  An integer, $listlen, where $listlen is less
#         than 100, followed by $listlen-integer values.
# Output: The number of input values that are greater than
#         the average of all input values.
($sum, $result) = (0, 0);
$listlen = <STDIN>;
if (($listlen > 0) && ($listlen < 100)) {
# Read input into an array and compute the sum
  for ($counter = 0; $counter < $listlen; $counter++) {
    $intlist[$counter] = <STDIN>;
  } #- end of for (counter ...
# Compute the average
  $average = $sum / $listlen;
# Count the input values that are > average
  foreach $num (@intlist) {
    if ($num > $average) { $result++; }
  } #- end of foreach $num ...
# Print result
  print "Number of vlues > average is: $result \n";
} #- end of if (($listlen ...
else {
  print "Error--input list length is not legal \n";
}
```

Prolog — 1972

- Aix-Marseille Üniversitesinde Comerauer ve Roussel tarafından geliştirildi. Edinburgh Üniversitesinden Kowalski yardım etti.
- Mantığa dayanır.
- Yordamsal değildir (Non-procedural).
- Akıllı veri tabanına dayalı, sorgulardan doğru sonuca olaşma olarak özetlenebilir.
- Prolog veri tabanı iki tip deyimden oluşur: olgular (facts) ve kurallar (rules).
- Faktöriyel hesaplama örneği verelim:

```
factorial(0,1). %% olgu: 0! = 1 dir.
factorial(N,F) :- N>0, %% kural: N! = N*(N-1)!
N1 is N-1, %% virgül 've' mantıksal işlecidir.
factorial(N1,F1),
F is N * F1.
```

- Tanımlamalar yukarıdaki şekilde yapılınca, aşağıdaki sorguda 'W' değişkeni sonucu döner:

```
?- factorial(3,W).
W=6
```

C++ - 1985 ve Java – 1995

- C++ (1985), Stroustrup tarafından Bell Labs'da geliştirildi.
  - Kısmen SIMULA 67'den alınan Nesne Tabanlı Programlama (NTP) özellikleri C'ye eklendi.
  - Büyük ve karmaşık bir dil.
  - NYP ile birlikte hızla popülerliği arttı
- Java (1995) Sun tarafından geliştirildi.
  - C++'a dayanır
  - Önemli ölçüde basitleştirildi
    - (C++ 'da bulunan struct, union, enum, ve atama zorlamalarının yarısı yoktur.)
  - Sadece NYP destekler.
  - Referanslar vardır fakat göstericiler (pointers) yoktur.
  - Dönemdeşliğe (concurrency) ve uygulamacıklara (applet) destek verir.

Java Kodu

```
// Java Example Program
//  Input: An integer, listlen, where listlen is less
//         than 100, followed by length-integer values
// Output: The number of input data that are greater than
//         the average of all input values
import java.io.*;
class IntSort {
public static void main(String args[]) throws IOException {
    DataInputStream in = new DataInputStream(System.in);
    int listlen,
        counter,
        sum = 0,
        average,
        result = 0;
    int[] intlist = new int[99];
    listlen = Integer.parseInt(in.readLine());
    if ((listlen > 0) && (listlen < 100)) {
  /* Read input into an array and compute the sum  */
      for (counter = 0; counter < listlen; counter++) {
        intlist[counter] =
              Integer.valueOf(in.readLine()).intValue();
        sum += intlist[counter];
      }
  /* Compute the average */
      average = sum / listlen;
  /* Count the input values that are > average */
      for (counter = 0; counter < listlen; counter++)
        if (intlist[counter] > average) result++;
  /* Print result */
        System.out.println(
            "\nNumber of values > average is:" + result);
      }  //** end of then clause of if ((listlen > 0) ...
      else System.out.println(
              "Error--input list length is not legal\n");
    }  //** end of method main
  }  //** end of class IntSort
```

Betik Dilleri (Scripting Languages)

- JavaScript (1985)
  - Çoğunlukla kullanıcı tarafı, HTML içinde gömülü, tarayıcıda çalışan betik dilidir.
  - Kullanıcı tarafında dinamik web dökümanlarında ve veri girişi kontrolünde çok kullanılır.
  - Tamamen yorumlanan bir dildir.
- PHP – 1994 - Rasmus Lerdorf
  - Sunucu tarafı, HTML içinde gömülü, betik dilidir.
  - Çoğunlukla web üzerinden form işleme veritabanı erişimleri için kullanılır.
  - Tamamen yorumlanan bir dildir.

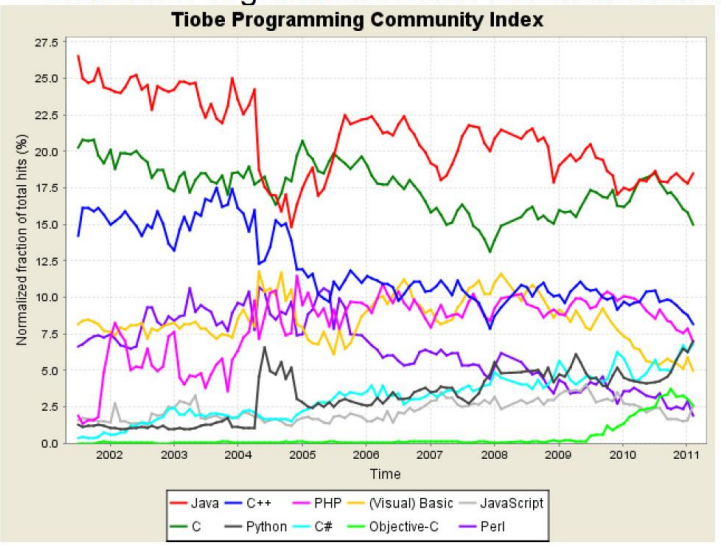
Betik Dilleri (Devam)

- Python –1990'lı yılların başları – Guido Van Rossum
  - Sistem yönetimi, CGI programlama
  - Tip kontrollü dinamik tipleme.
  - Dizilimler (array) yerine listeler, değiştirilemez listeler (tuples), and kıyımlı listeler (dictionaries)
- C# - 2000 - Microsoft
  - .NET platformları için ana dil.
  - Java ve C++ takipcisi.
  - Java'nın çoğu özelliğini bir kısım değişikliklerle ve bazı C++ özelliklerini kapsar.
  - web üzerinde .NET uygulamaları için kullanılabileceği gibi genel amaçla da kullanılabilir.

Bağlantılı metin/Programlama Melez Dilleri (Markup/Programming Hybrid Languages)

- XSLT
  - XML dokümanlarının ekranda gösterimi için kullanılır.
  - Etiketler şeklinde kontrol yapıları içerir, örneğin: <for-each>
- JSP
  - "Java Server Page" XHTML ve Java'nın karışımıdır.
  - Sayfalar JSP işlemcisi tarafından işlenerek sunucu java uygulamaları haline getirilir.
  - JSTL, JSP dokümanının işlemesini kontrol eden XML hareket elemanlarını tanımlar.
  - Örnek hareket elemanları: <if>, <forEach>, vs.

Günümüz Programlama Dilleri Kullanıcı Oranları



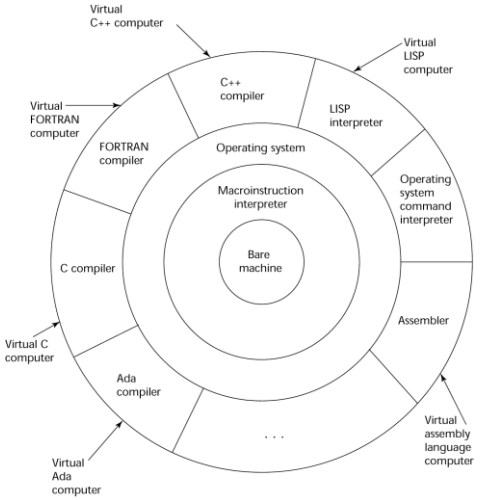
http://www.tiobe.com/index.php/content/paperinfo/tpci/

Gerçekleştirme Metotları

- Derleme(Compilation)
  - Programlar makine diline çevrilir.
- Saf yorumlama(Pure Interpretation)/Yorumlama
  - Programlar yorumlayıcı adlanan başka bir program tarafından yorumlanır
- Hibrit sistemler
  - Derleyici ile yorumlayıcının ortak kullanımı şeklindedir.

Bilgisayarın Katmanlı Görünümü

İşletim sistemi ve dil gerçekleştirimi, bir bilgisayarın makine arayüzünün üstünde katmanlanmıştır.



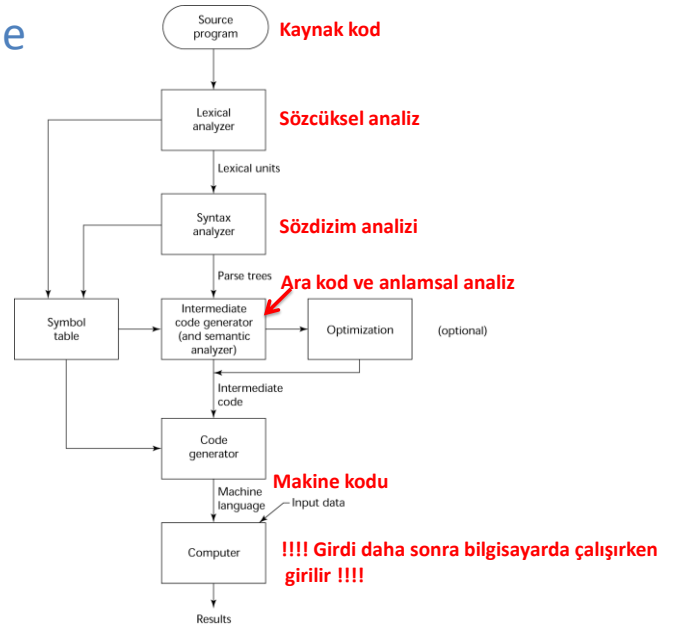
Derleyicilere karşı Yorumlayıcılar (Compilers Vs. Interpreters)

- Çeviri yürütmeden ayrı mı?
  - Evet – Derleyici (Compiler)
  - Hayır – Yorumlayıcı (Interpreter)
  - Hybrid sistemler

Derleme (Compiler)

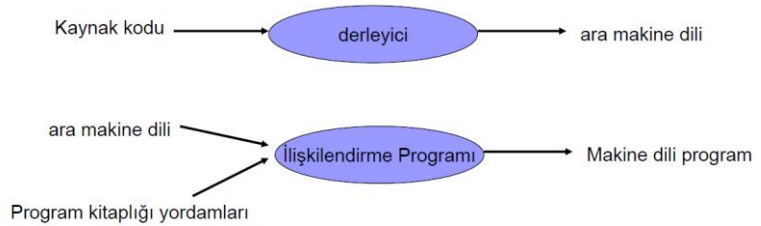
- Yüksek seviye program(kaynak dil/kod) makine koduna (makine dili/kodu) çevrilir.
- Yavaş çeviri, hızlı çalıştırma
- Derleme süreci birkaç safhadır:
  - Sözcüksel analiz(lexical analysis): kaynak programdaki karakterler sözcüksel birimlere dönüştürülür.
  - Sözdizim analizi(syntax analysis): sözcüksel birimler ayrıştırma ağaçlarına(parse tree) dönüştürülür. Bu ağaçlar programın söz dizim yapısını temsil eder.
  - Anlamsal analiz(Semantics analysis): Ara kod oluşturulur
  - Kod üretme: Makine kodu üretilir.

Derleme Süreci



Neden Derleyici (Compiler)?

- Temel mühendislik prensipleri
  - doğruluk – erken statik hata kontrolü
  - maliyet – derleme program dağıtma maliyetini düşürür.
  - performans – hızlı çalışır (derleme yavaş çevirse de daha sonra kod hızlı çalıştır)
    - bir defa derle (maliyet) , birçok defa yürüt (fayda)
- Yazılımın/fikirlerin korunması
  - Kaynak kodu derleyici ile ara makine diline çevrilir

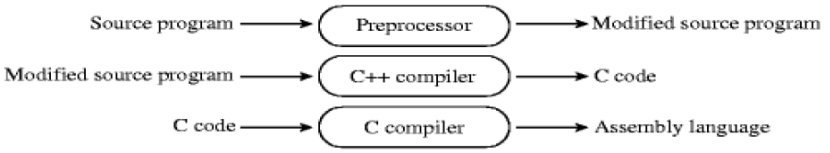


Çok Geçişli Derleyicilerin Kullanımı – Önişlemciler(Preprocessors)

- Karmaşıklığı nasıl çözümleyelim?
- Program kitaplığı ile (dili basit tutarak, örneğin C, Java).
- Ön işlemci makroları(talimatları) yaygın olarak başka bir dosya içindeki belirli kodun belirtilmesi için kullanılır.
- Önişlemci makroları işleyicisi program derlenmeden hemen önce bu makroları açar.
- İyi bilinen bir örnek: C önişlemci komutları #include, #define, ve benzer makrolardır.

Başka Dile Derleme

- Bazı derleyiciler çevirici diline derler (assembly code)
  - yüksek oranda optimize olur
- Bazı diğer derleyiciler başka üst seviye dile derleyebilir.
  - Var olan dilin optimizasyonunu kullanılır.
  - Taşınabilirliği artırır, karmaşıklığı azaltır.



Makine kodunun çalıştırılması

- Getir-çalıştır-döngüsü(Fetch-execute cycle) (von Neumann mimarisinde)

initialize the program counter **repeat** forever fetch the instruction pointed by the counter increment the counter decode the instruction execute the instruction **end repeat**	Program sayacını sıfırla **repeat** sonsuza kadar Sayacın gösterdiği talimatı getir Sayacı arttır Talimatın kodunu çöz Talimatı çalıştır **end repeat**

Von Neumann Darboğazı

- Bilgisayarın belleği ile işlemcisi arasındaki bağlantı hızı bilgisayarın hızını belirler
- Program talimatları genellikle bağlantı hızından daha hızlı çalıştırılırlar; böylece bağlantı hızı darboğaz oluşturur.
- Bu von Neumann darboğazı olarak bilinir; bu bilgisayarların hızını sınırlandıran birincil etkendir.

Saf Yorumlama (Pure Interpretation)

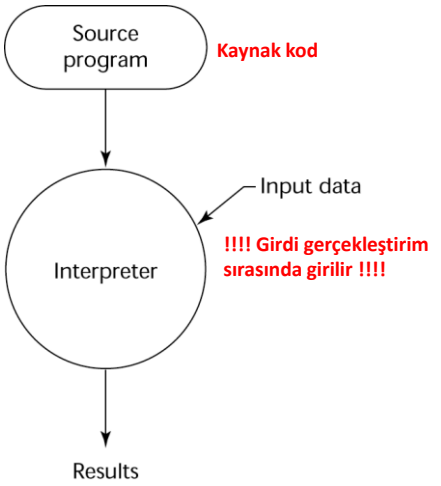
- Dönüştürme yoktur
- Programların gerçekleştirimi daha kolaydır (çalışma zamanı hataları kolaylıkla ve hemen görüntülenir.)
- Daha yavaş çalıştırma(derlenmiş programlardan 10 -100 kez daha yavaş çalışırlar)
- Genellikle daha fazla yere (belleğe) ihtiyaç duyarlar
- Yüksek seviye dillerde nadirdir
- Bazı Web betik dilleriyle yeniden geri gelmiştir.(örn., JavaScript)

Neden yorumlayıcı?

- Esneklik (yürütüm zamanı oluşan geç bağlantılar nedeniyle)
  - Yürütüm zamanı durum desteği
  - komut dosyaları (Perl, Shells, Python)
  - dinamik ortamlar (Basic)
  - sanal makineler (JVM, CPUs).

Saf Yorumlama Süreci

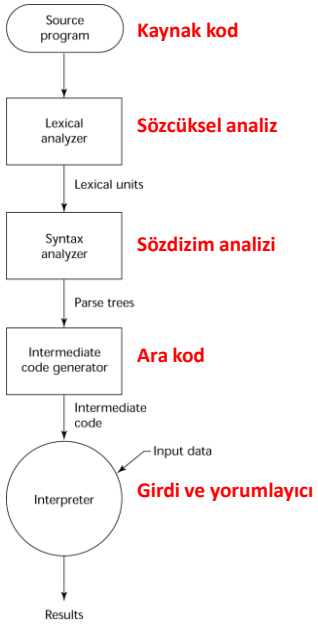
- Makine koduna dönüştürme yoktur



Hibrit (Hybrid) Gerçekleştirim Sistemleri

- Derleyiciler ile saf yorumlayıcıların ortak kullanımı
- Yüksek seviye dil bir **ara dile çevrilir**. Bu ara dilin yorumlanması daha kolaydır.
- Saf yorumlamaya göre daha hızlıdır.
- Örnekler
  - Perl programları yorumlamadan önce hataların tespiti için kısmi derlenirler.
  - Java da ara biçim byte code olarak adlanır. Bu arakod bir byte code yorumlayıcı ile yorumlanır ve çalıştırılır(JVM=Java virtual machine). JVM yorumlayıcısı olan her makine Java kodunu çalıştırabilir.

Hibrit Gerçekleştirim Süreci



Aynı Zamanda Gerçekleştirim(Just-in-Time (JIT) Implementation) Sistemleri

- Başlangıçta programlar ara dile çevrilir.
- Daha sonra ara diller makine kodlarına çevrilir.
- Makine kodu versiyonu daha sonraki çağrılmalar için saklanır.
- JIT sistemler Java programları için geniş şekilde kullanılmıştır (byte code).
- .NET dilleri JIT sistemler ile gerçekleştirilmektedir.

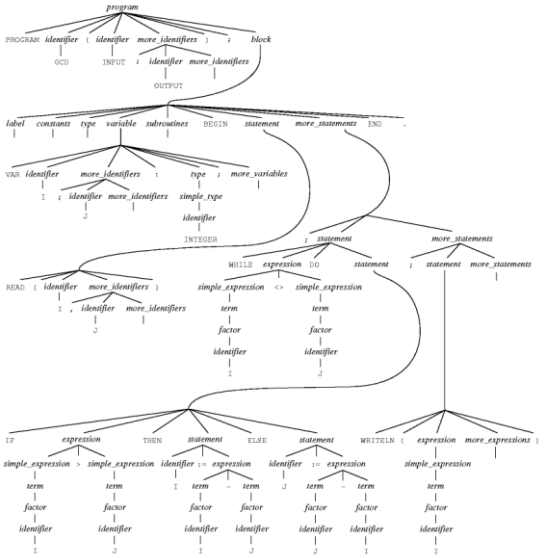
Bir Örnek

- Bir Pascal Programı düşünelim.

```
program gcd(input, output);
var i, j: integer;
begin
read(i, j);
while i <> j do
if i>j then i := i – j;
else i := j – i;
writeln(i);
end.
```

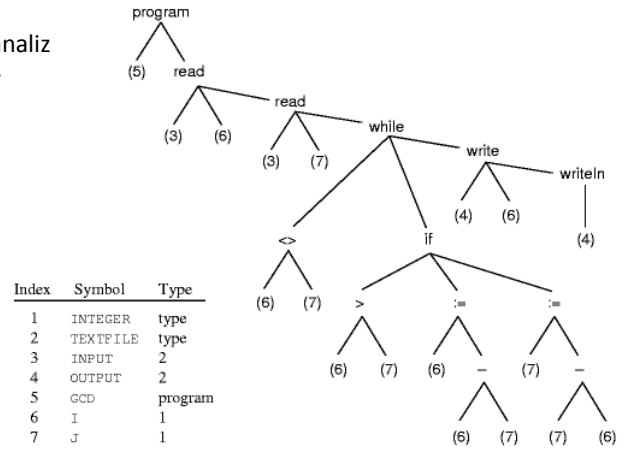
Sözdizim (Syntax) Analizi

- Tarama uçları belirler (tokens)
- Ayrıştırma uç olmayanları belirler.



Anlambilimsel (Semantic) Analiz

- Anlambilimsel analiz son kısımdır.
- Özet sözdizim analiz ağacı kullanırlır.



Index	Symbol	Type
1	INTEGER	type
2	TEXTFILE	type
3	INPUT	2
4	OUTPUT	2
5	GCD	program
6	I	1
7	J	1

Optimizasyon

- Hedef kaynak tüketimini azaltmak
  - bellek (veri veya kod)
  - yürütme zamanı

Programlama Çevreleri

- Yazılım geliştirme için kullanılan araçlar topluluğu
- UNIX
  - Eski bir işletim sistemi ve araçlar topluluğu
  - Bu günlerde GUI (örn, CDE, KDE, or GNOME) kullanılıyor
- Netbeans, eclipse
  - Entegre Java geliştirme ortamlarıdır
- Microsoft Visual Studio.NET
  - Büyük, karmaşık görsel geliştirme ortamıdır.
  - C#, Visual BASIC.NET, Jscript, C++ dilleri için kullanılır.

Özet

- Programlama dillerini çalışmanın önemi:
  - Farklı yapıları kullanma kapasitemizi arttırır.
  - Dilleri daha zekice seçmemizi sağlar.
  - Yeni dillerin öğrenilmesini kolaylaştırır.
- Bir programlama dilini değerlendirmede en etkin rol oynayan kriterler:
  - Okunabilirlik, yazılabilirlik, güvenilirlik, maliyet
- Dil geliştirmede en önemli etkiler makine mimarisi ile yazılım geliştirme metodojileridir.
- Bir programlama dilini gerçekleştirmedeki temel metotlar: derleme, sağ yorumlama ve hibrit gerçekleştirimdir.