**1.HAFTA**

**Programlama Dili**

Kısaca bilgisayara bir şeyler yaptırmamızı sağlayan araç olarak tanımlayabiliriz. Bu diller ile yazılımlar/programlar geliştirebiliyoruz.

**Not:** Bundan sonra yazılım/program yerine sadece yazılım denecek. (İkisi arasındaki fark araştırılacak)

**Programlama Dillerinin Amacı**

Hoca her ne kadar kendince amaçlar sıralamış olsa da bence hepsi saçma. Yazılanlar bir amaç olmaktan çok sanki bir tanım belirtiyor ki bu tanımların hepsi çok dolaylı adeta kulağa ters tutuyormuşçasına duruyorlar. Bence programlama dillerinin birçok amacı olabilir ki bunlar şu şekilde sıralanabilir:

* Anlaşılabilir olmak.
* Verimli olmak.
* Grup çalışmasına uygun olmak.

Fakat bence bütün dillerin ortak yegâne amacı şudur: "Bilgisayara komut vermemizi sağlamak". Diğer her şey, dili tasarlayanların bazı problemleri çözmek için koydukları amaçlardır. Bu problemlere örnek olarak şunlar verilebilir:

* Donanımı elverişli kullanmak.
* İşletim sistemi geliştirmek.
* Herkes tarafından rahatça öğrenebilmek.
* Hızlı yazılım geliştirmek.
* Aynı kodla birçok farklı sistemde çalışabilmek.
* ...

**Programlama Dillerinin Sınıflandırılması**

Programlama dillerinin birçok farklı şey için sınıflandırabiliriz. Biz bu derste 3 farklı şekilde sınıflandırıyoruz:

1. Seviyelerine Göre Sınıflandırma
2. Uygulama Alanlarına Göre Sınıflandırma
3. Tasarım Paradigmalarına Göre Sınıflandırma

**Seviyelerine Göre Sınıflandırma:**

* Makine Dili
* Alçak Seviyeli Diller (Sembolik Dil - Assembly)
* Orta Seviyeli Diller
  + Ada
  + C
* Yüksek Seviyeli Diller (3. Kuşak Dil)
  + Fortran
* Çok Yüksek Seviyeli Diller
  + Java
  + C#
  + Visual Basic
  + Access
  + Oracle Forms

Not: Açıkçası bu sınıflandırmayı tam olarak anlayamadım. Yani alçak seviyeli dil ile çok yüksek seviyeli dil arasındaki fark tam olarak ne? Bunları alçak veya çok yüksek yapan nedir? Sunumlarda yazan ifadeler çok muğlak. İnternetten baktığımda da benzer bir muğlaklık görüyorum. Makine Dili ve Alçak Seviyeli kavramları bir nebze oturuyor ama diğerlerinin arasındaki farklılık çok belirsiz. Orta-Yüksek-Çok Yüksek seviyeleri arasında anlamlı bir fark göremiyorum. Ayrıca bu sınıflandırma ile kuşaksal sınıflandırma da yapılıyor ki onda da aynı belirsizlik bulunuyor.

**Uygulama Alanlarına Göre Sınıflandırma:**

* Bilimsel ve Mühendislik Uygulama Dilleri
  + Pascal
  + C
  + Fortran
* Veri Tabanı Dilleri
  + MSSQL
  + Oracle Forms
  + XBASE
* Genel Amaçlı Programlama Dilleri
  + Pascal
  + C
  + Basic
  + Java
* Yapay Zekâ Dilleri
  + Prolog
  + Lisp
* Modelleme Yapmak Üzere Geliştirilen Simülasyon Dilleri
  + GPSS
  + Simula67
* Makro Diller (Script Diller)
  + awk
  + Perl
  + Python
  + Tcl
  + Javascript
* Sistem Programlama Dilleri
  + C
* Ticari Uygulamalara Yönelik Programlama Dilleri
  + Cobol

**Not:** Bu sınıflandırmadaki "Makro Diller" bölümü saçma. Bu bir uygulama alanı değil ki. Hoca sınavda sorar diye yazdım.

**Tasarım Paradigmalarına Göre Sınıflandırma:**

* Emir Esaslı Programlama
  + C
  + Fortran
  + Pascal
  + Cobol
* Nesneye Yönelik Programlama
  + Simula67
* Fonksiyonel Programlama
  + Lisp
  + Scheme
  + ML
* Mantık Esaslı Programlama

Sunumda verilenler dışında birçok programlama paradigması bulunmaktadır. Hatta bazıları birbirinin türevidir. Örneğin:

* Emir Esaslı (Imperative)
  + Prosedürel (Procedural)
    - C
  + Nesneye Dayalı (Object Oriented)
    - C++
    - C#
    - Java
* Bildirimsel (Declarative)
  + Fonksiyonel (Functional)
    - Lisp
  + Mantık (Logic)
    - Prolog
* Yapısal (Structured): Bu paradigmaya sahip diller if/else gibi kontrol yapıları, for/while gibi döngü yapıları ve fonksiyon çağrımlarının hepsini içerir.

**Programlama Dillerinin Değerlendirme Ölçütleri**

Programlama dilleri ile bilgisayara birçok şey yaptırabiliyoruz. Yapmaya çalıştığımız şeyler için programlama dillerinden farklı şeyler isteyebiliyoruz. Mesela bir işletim sistemi yapmayı amaçlayan ile bir hastane yönetim yazılımı yapmak isteyen kişilerin programlama dilinden beklentileri farklı olabiliyor. Bundan dolayı programlama dillerini yapacağımız iş için değerlendirmeliyiz. Fakat bu değerlendirme için elimizde ölçütler (kriterler) olmalı. Peki nedir bu ölçütler? İşletim sistemi geliştirebilmek bile bir ölçüt olarak alınabilir fakat bu çok geniş bir tanıma sahip bir ölçüt olur. Bundan kaçınmak için bu tür geniş tanımlı isteklerimizi küçük parçalara bölmeliyiz. Bu tür geniş tanımlı ölçütleri küçük parçalara ayırdığımızda ortak şeyler olduğunu görüyoruz. Derste bunlara örnek olarak şunlar verilebilir.

* İfade Gücü (Expression Power)
* Veri Türleri ve Yapıları (Data Types and Structures)
* Giriş/Çıkış Kolaylığı (Input/Output Facilities)
* Taşınabilirlik (Portability)
* Alt programlama Yeteneği (Modularity)
* Verimlilik (Efficiency)
* Okunabilirlik (Readability), Yazılabilirlik
* Esneklik (Flexibility)
* Öğrenme Kolaylığı (Pedagogy)
* Genel Amaçlılık (Generality)
* Yapısallık (Structrulness)
* Nesne yönelimlilik (Object Orientation)

**2.HAFTA**

**Programlama Dillerinin Tarihçesi**

* Fortran
  + FORmul TRANslating System
  + 1954
  + IBM
  + Fortran 1 - Fortran 66 - Fortran 77 - Fortran 90 - Fortran 95
* Lisp
  + 1958
  + Fonksiyonel
  + Yapay Zeka
  + Atom ve Liste
  + Scheme ve Common Lisp
* Algol
  + ALGOrithmic Language
  + 1958
  + Fortran'dan esinlendi.
  + Algol 58 - Algol 60 - Algol 68
* Cobol
  + Common Business Oriented Language
  + 1959
  + İşletmelere yönelik
* Basic
  + 1964
  + Kolay
  + Eğitim amaçlı
* PL/I
  + Programming Language One
  + 1965
* Simula67
  + Simulasyon için tasarlandı.
  + İlk nesne yönelimli dil
* Pascal
  + 1971
  + Kolay
  + Eğitim amaçlı
* C
  + 1972
  + Sistem programlama dili
* Modula
* Prolog
  + 1972
  + Yapay zeka
  + Kurallar ve Önermeler
* Ada
  + ABD savunma bakanlığı çalışması sonucu geliştirdi.
* Smaltalk
  + 1970
  + Tamamen nesne yönelimli
* C++
  + Hem emir esaslı hem de nesne yönelimli
* Eiffel
  + Hem emir esaslı hem de nesne yönelimli
  + C++'a benzer
* Delphi
  + Hem emir esaslı hem de nesne yönelimli
  + Pascal'dan esinlenildi.
* Java
  + Tamamen nesne yönelimli
  + Çöp toplayıcı
* Javascript
  + İstemci taraflı betik (script) bir dildir.
* Php
  + Sunucu taraflı betik (script) bir dildir.
* Python
  + 1991
  + Betik (script) bir dildir.
  + Çöp toplayıcı
* C#
  + CLR
  + .NET Framework

**Ders Sonu Kodlama Bölümü**

1) Java'da karakterler 16 bit (Unicode) olarak C'de 8 bit (ANSII) olarak saklanmaktadır.

2) Java dahili olarak "boolean" değişken tipini destekler. Fakat C'de böyle değildir. C'ye göre 0 olmayan bütün değerler "doğru" olarak kabul edilir. "bool" ifadesi kullanılmak isteniyorsa ya "stdbool.h" şeklinde bir sınıf tanımlanmalıdır ya da şu şekilde bir tanımlama yapılmalıdır:

|  |  |
| --- | --- |
| 1  2  3  4  5 | **typedef** **enum**  {  false,  true  } **bool**; |

3) Java'da tanımlanan veri tipleri bilgisayardan bilgisayara göre değişmez. C'de ise kullanılan derleyiciye göre değişebilir.

4) Her iki dilde de "float" ve "double" olarak tanımlanan verilere aynı değerler verilse de aynı değillerdir. Çünkü bu iki veri türünün farklı hassasiyetleri vardır.

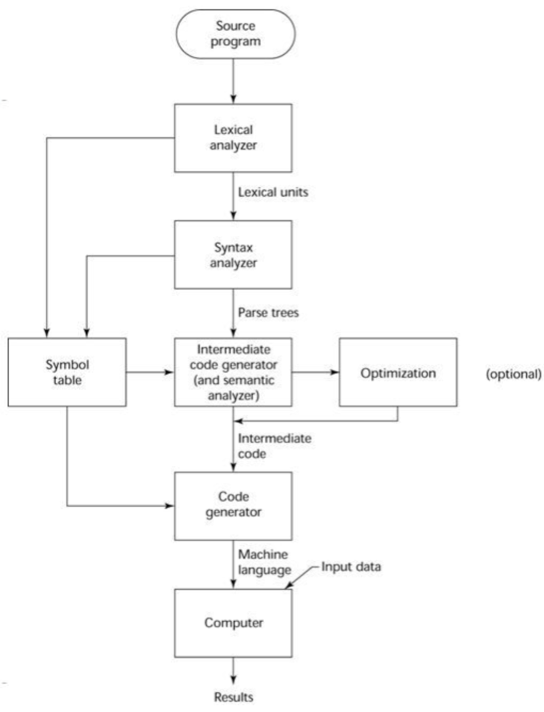
5) Java'da sabit bir değişken tanımlamak için "final" ifadesi kullanılır. C'de ise "const" ifadesi kullanılır. Ayrıcı Java'da başlangıç değeri verilmeyen sabit değere sonradan değer verilebilir.

6) Java "var" ifadesini destekler.

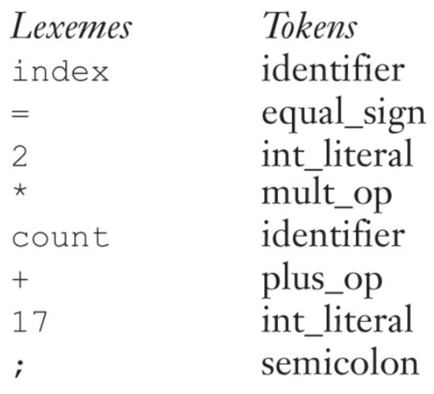
**3.HAFTA**

**Programlama Dilinin Derlenme Süreci:**

Dildeki anlamlı en küçük yapıya **lexeme** denir. Bir nevi doğal dillerdeki kelimelere benzetebiliriz. Bu yapıları farklı kategorilere ayırabiliriz. Bu kategorilere de **token** denir. Örneğin:

* Değişken isimleri - Identifier
* Dildeki anahtar kelimeler - Keyword (if, while, ...)
* Dilde kullanılan noktalama işaretleri - Seperator (Mesela C ve C++'daki ";" işareti)
* Operatörler - Operator(<, =, >, >=, +, /, ...)
* Değişmez ifadeler - Literal (Sayılar, "asd" gibi stringler)
* Yorum ifadeleri - Comment

index = 2\*count+17;



Dil derlemesi işleminin daha basit ve verimli olması için bu işlem Lexical Analysis, Syntax Analysis ve Semantic Analysis olmak üzere 3 ayrılır.

**Lexical Analysis (Scanner):** Bu bölüme geçilmeden önce önişlemci (preprocessor) komutlarda yazılan şeyler yerine getirilir. Derleme sürecinde kaynak kodun lexeme'lare ve token'lara ayrıldığı kısımdır. Dile ait olmayan token yapısıyla karşılaşılınca hata verir. Bu sistem düzenli ifadeleri (Ragular Expression) kullanan bir pattern matcher'dır. Bu kısımdaki analize hem doğal dil hem de programlama dili açısından örnekler verebiliriz:

Konuştuğumuz Dil: Ben giddim. ---> Bu cümlede "giddim" kelimesi yanlış yazılmış. Doğrusu şöyle olmalıydı: "Ben gittim."

Programlama Dili (C++): int 3degisken = 0; ---> Dikkat ederseniz değişken ismi C++ isim kurallarına göre hatalı. Bir değişken ismi, bir karakter veya "\_" başlamalıdır. Bundan dolayı ifade şöyle olmalıydı: int degisken = 0;

**Syntax Analysis (Parser):** Kaynak kod üzerinde sözdizimi analizi yapılan bölümdür. Lexeme'lere ayrılan kaynak koddaki lexeme'ler arasındaki ilişkileri inceler. Konuştuğumuz diller kelimelerden oluşur ve kelimeler cümleleri oluşturur. Lexeme'ler bu kelimeler gibidir. Lexeme'ler bir araya gelerek komutları oluşturur. Bu yapılarda dilimizdeki cümlelere benzer. Bu analiz cümlelerin doğru şekilde kurulup kurulmadığını inceler. Örneğin:

Konuştuğumuz Dil: Ben gideceğim okula. ---> Bu cümlede göreceğiniz üzere sözcükler gayet doğrudur fakat sözcüklerin dizilimi yanlıştır. Doğrusu "Ben okula gideceğim" olmalıydı.

Programlama Dili (C++): int degisken 5 =; ---> Dikkat ederseniz aslında burada bir değişkene atama yapılmak istenmiş fakat lexeme'ler yanlış yerleştirilmiştir. Aslında şöyle olmalıydı: int degisken = 5;

Bu bölümde bu ayrıştırmaları yapabilmek için ayrıştırma ağaçları (parse tree) oluşturulur. Ayrıştırma ağacına örnek olarak şöyle bir şey örnek verilebilir:

metin, dış mekan, saat, işaret içeren bir resim

Açıklama otomatik olarak oluşturulduYandaki matematiksel işlemlerde farklı notasyonlar kullanılabiliyor. Şunlar şöyledir:

1. Infix Notation: A + B

2. Prefix Notation: + A B

3. Postfix Notation: A B +

Bilhassa bu tür ifadeleri yığın (stack) yapısıyla rahat bir şekilde çözmemizi sağlayan postfix notation bu işlerde çok kullanışlıdır.

Programlama dillerinde sözdizimi analizi yapabilmek için bu dillerin bir gramer yapısına ihtiyaçları vardır. İnsanoğlu dillerin bu gramer yapılarını göstermek için birbirinden farklı birçok araç geliştirdi. Bunlardan bazıları CFG, BNF ve EBNF'dir.

**BNF (Backus - Naur Form):**

Bu gramer aracında 2 adet yapı bulunur: Terminal ve Nonterminal. Terminaller lexeme veya token olabilir. Nonterminal ise terminal olmayan her şeydir. Nonterminal'ler elinde sonunda terminal'lere indirgenebilir olmalıdır. Örnek:

<assign> -> <var> = <integer> | <var> = <float>

Yukarıdaki örnekte "<" ile ">" işaretleri arasında olanlar nonterminal'leri ifade eder. Bu işaretlerin arasında olmayanlar terminal'lerdir. "->" ifadesi bize "anlamına gelir" demek ister. "|" işareti "veya" anlamına gelir.

Bu örnek bize şunu demektir: "assign", "var" = "integer" veya "var" = "float" anlamına gelir.

Bunu şu şekilde de diyebiliriz: Bir atama işlemi "değişken = tam sayi" veya "değişken = ondalıklı sayı" şeklinde ifade edilebilir.

Bu ifade tek başına bir gramer olarak kabul edemeyiz. Çünkü "<var>", "<integer>" ve "<float>" ifadelerini ne olduğunu açıklamalıyız. Yani bir terminal'lere indirgemeliyiz. Unutulmamalıdır ki bütün nonterminal'ler bir terminal'e ya da terminal'ler bütününe indirgenmez ise gramer elde etmiş olamayız. Yukarıda verdiğimiz örneği bir gramere şöyle çevirebiliriz:

|  |
| --- |
| stmts: Program komutları  stmt: Program komutu  var: variable  expr: expression  const: constant (Sabit sayı: 1,42) |

<program> ---> <stmts>

<stmts> ---> <stmt> | <stmt> ; <stmts>

<stmt> ---> <var> = <expr>

<var> ---> a | b | c | d

<expr> ---> <term> + <term> | <term> - <term>

<term> ---> <var> | const

Bu gramere göre türetimler şu şekilde yapılır:

<program> ---> <stmts> ---> <stmt>

--> <var> = <expr>

--> a = <expr>

--> a = <term> + <term>

--> a = b + <term>

--> a= b + constant

Bu gramere göre şöyle şeyler yazılabilir:

• a = 5

• b = a + 4,5

• a = a + 3

• b = 3,6 + b

Gramere bir başka örnek daha vermek istersek şöyle bir şey diyebiliriz:

<assign> ---> <id> = <expr>

<id> ---> A | B | C

<expr> ---> <id> + <expr> | <id> \* <expr> | ( <expr> ) | <id>

Bu gramere göre şöyle bir türetim yapabiliriz:

<assign> ---> <id> = <expr>

--> A = <expr>

--> A = <id> \* <expr>

--> A = B \* <expr>

--> A = B \* ( <expr> )

--> A = B \* ( <id> + <expr> )

--> A = B \* ( A + <expr> )

--> A = B \* ( A + <id> )

--> A = B \* ( A + C )

1) Diziler

2) String

3) Pointer

4) Bellek bölgeleri

5) void ve object

6) veri tiplerinde işlemler

7) işlem önceliği