: B B M 3 0 1 - P R O G R A M M I N G L A N G U A G E S :

1) *DERS 01 - GİRİŞ:*

- 702 Programlama dili mevcut.
- İlk program ENIAC.
- Programlama Dilleri Kavramlarının Çalışılma Nedenleri; (**http://cs.loc.edu/~chu/COSI350/Ch1/Reason.html**)
  - ✓ **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.*)
  - ✓ **Belli bir dilin önemli özelliklerini anlayarak daha iyi kullanabilmek için.** (*Diller komplex yapılardan olusur. Fakat önemli özellikler etkin kullanılarak yazılım gelistirilebilir.*)
  - ✓ **Bir kod yazabilir ve derleyicinin her şeyi yapmasını sağlayabiliriz, ancak uygulama ayrıntılarını bilmek bir dili daha akıllı bir şekilde kullanmamıza ve daha verimli bir kod yazmamıza yardımcı olur.**
  - ✓ **Dilleri daha iyi değerlendirebilirsek, doğru seçimler yaparız, doğru teknolojilerin gelişmesine destek olmuş oluruz.**
  - ✓ **Gerçekleştirimin anlaşılmasıyla programlama dilini daha iyi anlama.** (*Örnek: Alt programlar sıklıkla çağrılırsa, program hızı düşer. Bunu bilirsek daha iyi program tasarımı yapabiliriz.*)
  - ✓ **Hata bulurken özelliklerini bilmemiz faydalıdır.**
  - ✓ **Tıpkı doğal diller gibi, bir dilin gramerini ne kadar iyi biliyorsak, ikinci bir dil öğrenmek o kadar kolay olacaktır.**
  - ✓ Programlama dilleri kavramlarını inceleyerek, programcılar dillerin önceden bilmedikleri kısımlarını kolayca öğrenebilirler.
- Dil Değerlendirme Kriterleri; [**Readibility**/**Writability**/**Reliability**/**Cost**] (**http://ece.uprm.edu/~ahchinaei/courses/2010jan/icom4036/slides/03icom4036Intro.pdf**)
  - ✓ **Okunabilirlik;**
    - ➢ Programlar kolay okunabilir ve anlaşılır olmalı.
    - ➢ **Bütünün Basitliği;** Yönetilebilir özellikler ve yapılar, Aynı işi yapan özelliklerin çok olması (• Örnek: c = c + 1; c+ = 1; c++; ++c;)
    - ➢ **Orthogonality**; İlkel yapıların küçük sayıdaki yollar ile bir araya getirilerek birleştirilebilmesi (• Örnek: 4 veri tipi: integer, float, double, char ve 2 tip operatör: dizi ve pointer kullanarak birçok kontrol ifadesi yazılabilir.), Birbirinden bağımsız yapıların varlığı ve tanımlanması. (Pascal ve C ortogonal değildir.)

- **Kontrol ifadeleri;** İyi bilinen kontrol ifadelerinin varlığı (örn., while ifadesi)
- **Veri Tipleri ve Yapıları;** Veri yapılarını 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)

Orthogonality

- **Example :** Adding two 32-bit integers residing in memory or registers, and replacing one of them with the sum
- IBM (Mainframe) Assembly language has two instructions:

```
A   Register1, MemoryCell1
AR  Register1, Register2
```

More restricted
Less writable
Not orthogonal

meaning

Register1 ← contents(Register1) + contents(MemoryCell1)
Register1 ← contents(Register1) + contents(Register2)

- VAX Assembly language has one instruction:

```
ADDL operand1, operand2
```

orthogonal

meaning

operand2 ← contents(operand1) + contents(operand2)
Here, either operand can be a register or a memory cell.

✓ **Yazılabilirlik;**

- Program oluşturmak için yazımının kolay olması.
- **Basitlik;** Az yapının olması, küçük sayıda ilkellerin olması, bunları birleştirecek kuralların az olması.
- **Soyutlama Desteği;** Detayları yok sayarak karmaşık yapı ve işlemleri tanımlama ve kullanma yeteneği
- **Anlamlılık;** İşlemleri tanımlamak için uygun yolların olması, (Örnek: for ifadesinin yerine daha kolay yazılabilen while'ı kullanmak)
- **Okunabilirlik ve Yazılabilirlik;** Bir algoritmayı doğal bir şekilde ifade etme yolları bulunmayan diller, ister istemez doğal olmayan yaklaşımları kullanacaktır, böylece de okunabilirlik azalacaktır.

✓ **Güvenilebilirlik;**

- Teknik şartnamelere uygunluğu, tanımlara uyması.
  - **Tip Kontrolü;** Tip hataları için test etme. (Örneğin; yandaki uygulama tip kontrolü olmayan C programında çalışır. >>>>>>>>>>)
  - **İstisna (Exception) İşleme;** Çalışma zamanı hatalarının kesilmesi ve düzeltici önlemlerin alınması.
  - **Örtüşme (Aliasing);** Aynı bellek bölgesini işaret eden iki yada daha fazla farklı referansın olabilmesi iyi değildir.

For example, the following program in original C compiles and runs!

```
foo  (float  a)  {
  printf  ("a:  %g  and  square(a):  %g\n",  a,a*a);
}
main  ()  {
  char  z  =  'b';
  foo(z);
}
```

Output is : a: 98 and square(a): 9604

- ✓ **Maliyet;**
  - ➢ Dili kullanmak için programcıların eğitimi.
  - ➢ Program yazma maliyeti (özel uygulamalara kapalılık).
  - ➢ Programları derleme maliyeti.
  - ➢ Programları yürütme maliyeti.
  - ➢ Uygulama Sisteminin Maliyeti: Eğer program pahalıysa veya sadece pahalı donanımlarda çalışıyorsa, yaygın olarak kullanılmayacaktır.
  - ➢ Zayıf güvenilirlik yüksek maliyetlere neden olur.
  - ➢ Programların bakımı sonucu oluşan maliyet.
- ✓ **Diğer Kriterler;**
  - ➢ **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: (**https://www.slideshare.net/abreslav/trade-offs-22989326**)

- ✓ **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.

❖ Programlama Alanları: [**Scientific Applications**/**Business Applications**/**Artificial Intelligence**/**Systems Prog.**/**Scripting Languages**] (**http://cs.loc.edu/~chu/COSI350/Ch1/Domain.html**)

- ✓ **Bilimsel Uygulamalar;**
  - ➢ Büyük sayıda noktalı hesaplama yapma
  - ➢ Doğru hesaplama en önemli özellik
  - ➢ Fortran (1950'ler), Algol 60 (1960'lar)
- ✓ **İş Uygulamaları;**
  - ➢ Rapor oluşturma, ondalık sayı ve karakterlerin kullanımı
  - ➢ COBOL (1960'lar) halen en popüleri
- ✓ **Yapay Zeka;**
  - ➢ Sayılar yerine semboller kullanılır, dizi yerine bağlantılı bilgi
  - ➢ Programlar çok daha esnek yapıya sahip olmalıdır; program çalışırken yeni kod üretip çalıştırabilmelidir.
  - ➢ LISP (1965), Prolog (1970'ler)
- ✓ **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++ ileyazılmıştır.
  - ➢ UNIX tamamen C ile yazılmıştır.
- ✓ **Web Yazılımı;**

- ➢ Markup (örn. HTML, XHTML) bir programlama dili degildir.
- ➢ Scripting Languages (örn., PHP, Javascript) dinamik icerik icin HTML dökümanına program kodları eklemek için
- ➢ Genel Amaçlı (örn. Java (applets, servlets))

❖ Dil Kategorileri: [**Imperative**/**Functional**/**Logic**/**Object Oriented**/**Markup**] (**http://www.info.univ-angers.fr/~gh/hilapr/langlist/classif.htm**)

✓ **Emirsel;**

- ➢ Merkezi özellikleri değişkenler, atama ifadeleri ve döngülerdir. (Örnek: C, Pascal)

✓ **Fonksiyonel;**

- ➢ Hesaplama yapmanın temelinde veriler ve parametrele fonksiyonları uygulamak. (Örnek: LISP, Scheme)

✓ **Mantık;**

- ➢ Kural tabanlı, kurallar belirli sıralama olmadan verilir. (Örnek: Prolog)

✓ **Nesneye Yönelik;**

- ➢ Veri soyutlama, kalıtım, polymorphism. (Örnek: Java, C++)

✓ **İşaretleme;**

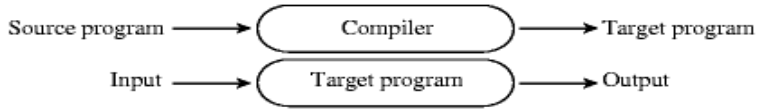
- ➢ Yeni; tam bir programlama dili değillerdir fakat web dökümanlarındaki bilginin yerleşimini belirtmede kullanılır. (Örnek: XHTML, XML)

2) DERS 02 - DESCRABING SYNTAX AND SEMANTICS:

2.1) Uygulama Yöntemleri: [Compilation/Pure Interpretation/Hybrid Implementation Systems]

❖ **Derleme;**

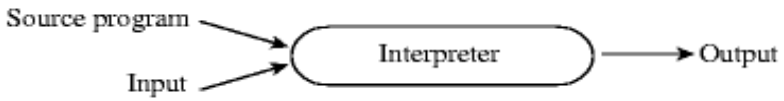
✓ Programlar makine diline çevrilir; JIT sistemleri içerir. (Kullanım: Büyük ticari uygulamalar) Yavaş Çeviri, Hızlı Çalıştırma !!! (C, C++, Cobol, Ada)



❖ **Hibrit Uygulama Sistemleri;**

❖ **Saf Yorumlama;**

✓ Programlar tercüman olarak bilinen başka bir program tarafından yorumlanır. (Kullanım: Küçük programlar veya verimlilik bir sorun olmadığında) Çeviri Yok, Yavaş Çalıştırma !!! (Script Dilleri)



- ✓ Derleyiciler ve saf tercümanlar arasında bir uzlaşma. (Kullanım: Verimlilik ilk sorun değilse küçük ve orta sistemler) Saf Yorumlama'dan Daha Hızlı !!!

2.2) Syntax (Söz Dizimi/Gramer) ve Semantics (Programlama Dilinin Davranışları):

❖ Bilgisayar Programları Oluşturma;

- ✓ Her programlama dili, **bir dizi ilkel işlem** sağlar.
- ✓ Her programlama dili, **karmaşık olan ilkel ifadeleri**, yasal bir biçimde birleştiren mekanizmalar sağlar.
- ✓ Her programlama dili, hesaplamalar veya ifadeler ile ilişkili **anlamların veya değerlerin çıkarılması** için mekanizmalar sağlar.

❖ Terminology;

- ✓ **Syntax,** ifadelerin ve program birimlerinin biçimi veya yapısı. (Örneğin; while)
- ✓ **Sematics,** ifadelerin ve program birimlerinin anlamı. (Örneğin; while'ın iç değerinin anlamı)
- ✓ **Sentence,** bazı alfabelerin bulunduğu söz dizisi.
- ✓ **Language,** cümle kümesi.
- ✓ **Lexeme,** bir dilin en düşük düzeyde sözdizimsel birimi (Örneğin; sum, begin)
- ✓ **Token,** Lexeme'nin kategorisi. (Örneğin; identifier)

Example in Java Language

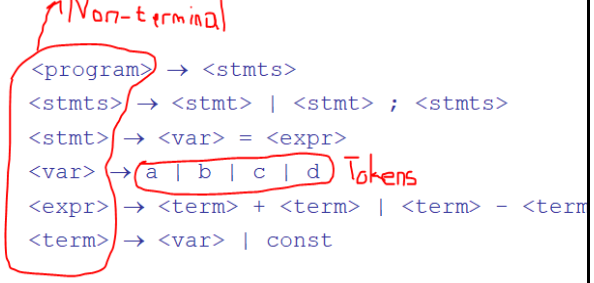
```
x = (y+3.1) * z_5 ;
```

```
x = (y+3.1) * z_5;
```

Lexemes	Tokens
x	identifier
=	equal_sign
(	left_paren
)	right_paren
for	for
y	identifier
+	plus_op
3.1	float_literal
*	mult_op
z_5	identifier
;	semi_colon

❖ BNF (Backus-Naur Form (1959))

- ✓ BNF'de soyutlamalar, sözdizimsel yapıların sınıflarını temsil etmek için kullanılır, sözdizimsel değişkenler gibi davranırlar (aynı zamanda nonterminal semboller veya sadece terminaller olarak da adlandırılır)
- ✓ Terminaller lexeme'ler ya da token'lardır.
- ✓ <LHS> → <RHS> >>>> Soldaki ifade için LHS; Non-terminal, RHS; Terminal dizisi ya da non-terminal ifade içerir.
- ✓ Sağda bir örneğini göreceğimiz "BNF" açılımında anlamlandırmak kolay olacaktır.




```
<program> => <stmts> => <stmt>
          => <var> = <expr>
          => a = <expr>
          => a = <term> + <term>
          => a = <var> + <term>
          => a = b + <term>
          => a = b + const
```

- ✓ ***Türetme (Derivation);***

- Verilen bir dizenin dilde geçerli bir programı temsil edip etmediğini kontrol etmek için, bunu dilbilgisinde türetmeye çalışırız.

- En soldaki türetme, her bir cümle içindeki en soldaki nonterminalin genişletilmiş olanıdır.

Derive string: **begin A := B; C := A * B end**

<program> ⇒ begin <stmt_list> end
⇒ begin <stmt> ; <stmt_list> end
⇒ begin <var> := <expression>; <stmt_list> end
⇒ begin A := <expression>; <stmt_list> end
⇒ begin A := B; <stmt_list> end
⇒ begin A := B; <stmt> end
⇒ begin A := B; <var> := <expression> end
⇒ begin A := B; C := <expression> end
⇒ begin A := B; C := <var><arith_op><var> end
⇒ begin A := B; C := A <arith_op> <var> end
⇒ begin A := B; C := A * <var> end
⇒ begin A := B; C := A * B end

sentential form

If always the leftmost nonterminal is replaced, then it is called **leftmost derivation**.

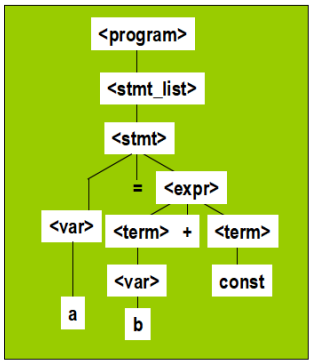
- Bir türetmedeki her sembol dizesi bir cümle biçimidir.

- Bir cümle, yalnızca terminal sembollerine sahip olan bir cümle biçimidir.

✓ ***Parse Tree;***

- A hierarchical representation of a derivation

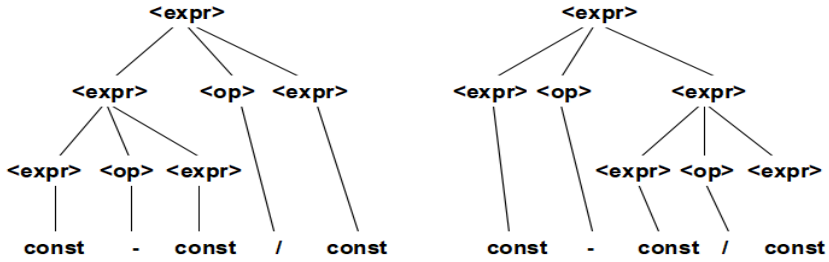
<program> → <stmt_list>
<stmt_list> → <stmt>
| <stmt> ; <stmt_list>
<stmt> → <var> = <expr>
<var> → a | b | c | d
<expr> → <term> + <term>
| <term> - <term>
<term> → <var> | const



✓ ***Belirsizlik (Ambiguous);***

- Bir dilbilgisi, yalnızca iki veya daha fazla ayrıştırma ağacı içeren bir cümle formu oluşturuyorsa belirsizdir.

<expr> → <expr> <op> <expr> | const
<op> → / | -



❖ ***Öncelik Kuralı (Precedence);***

✓ Bir ifadede birden fazla operatör varsa, C dili, işleçler için önceden tanımlanmış bir öncelik kuralına sahiptir. Bu işleçlerin önceliği kuralına "operatör önceliği" denir.

✓ C'de, aritmetik işleçlerin (*,%, /, +, -) önceliği ilişkisel işleçlerden (==,! =,>, <,> =, <=) daha yüksektir. İlişkisel işleçlerin önceliği de mantıksal işleçlerden daha yüksektir. (&&, ||).

```
(1 > 2 + 3 && 4)
This expression is equivalent to:
((1 > (2 + 3)) && 4)
i.e, (2 + 3) executes first resulting into 5
then, first part of the expression (1 > 5) executes resulting into 0 (false)
then, (0 && 4) executes resulting into 0 (false)
```

❖ ***Birleşme Kuralı (Associativity;***

- ✓ Bir ifadede aynı öncelikli (priority) iki operatör varsa, işleçlerin ilişkilendirilmesi yürütme sırasını gösterir.
- ✓ Burada, operatörler == ve! = Aynı önceliğe sahiptirler. Her ikisi de == ve! = Birlikteliği sağa sola, yani soldaki ifade ilk önce yürütülür ve sağa doğru hareket eder.

```
1 == 2 != 3
```

```
((1 == 2) != 3)
i.e, (1 == 2) executes first resulting into 0 (false)
then, (0 != 3) executes resulting into 1 (true)
```

❖ ***Extended BNF;***

- ✓ Bilgisayar bilimlerinde dil tasarımı konusunda kullanılan Backus Normal Şeklinin (BNF) özel bir halidir. Basitçe standart BNF'te yazılan kuralların birleştirilerek daha sade yazılmasını hedefler.
- ✓ Örneğin BNF olarak yazılan dilimize göre:
  - ➢ ***<IF> ::= if(<KOSUL>) | if(<KOSUL>) else,***
  - ➢ şeklinde bir satırımız bulunsun. Bu satırın anlamı dilimizde bir IF söz dizilimi (syntax), if komutu ve parantez içinde bir koşuldan oluşabilir veya bu if ve parantez içerisindeki koşulu bir else komutu izleyebilir.
  - ➢ Yukarıdaki bu BNF yazılımını EBNF olarak aşağıdaki şekilde yazabiliriz:
  - ➢ ***<IF> ::= if(<KOSUL>) [else]***
  - ➢ Yukarıdaki bu yeni satırda dikkat edileceği üzere köşeli parantezler arasında bir else komutu bulunmaktadır. Bunun anlamı, IF komutu "if(KOSUL)" olarak tanımlanır ve şayet istenirse bu komuta ilave olarak else komutu eklenebilir. Yani köşeli parantez içerisindeki komut isteğe bağlıdır.
- ✓ Yukarıdaki bu yeni yazılım aslında sadece gösterimde bir farklılık oluşturmaktadır. Bunun dışında, EBNF'in kullanım alanı ve işlevi BNF ile aynıdır.
- ✓ EBNF'in, BNF'den farkı yandaki tabloda işaretlenmiştir.

İfade		Kullanımı
Tanımlama	definition	=
Üleştirme	concatenation	,
Bitirme	termination	;
Seçim (Veya)	separation	|
Çift Tırnak	double quotation marks	" ... "
Tek Tırnak	single quotation marks	' ... '
İsteğe bağlı	option	[...]
Tekrarlı	repetition	{ ... }
Gruplama	grouping	(...)
Yorum	comment	(* ... *)
Özel dizilim	special sequence	? ... ?
Hariç	exception	-

- ✓ **<YORUM> ::= "/*" , { <harf> } , "*/"**
- ✓ **<harf> ::= a | b | … | z**
- ✓ Yukarıdaki EBNF tanımında a'dan z'ye kadar olan harfler, <harf> olarak tanımlanmış, ardından bu tanım <YORUM> içerisinde istenildiği kadar tekrarlanabilir anlamında {} işaretleri arasına yerleştirilmiştir.

3) DERS 03 - LEX: (http://bilgisayarkavramlari.sadievrenseker.com/2008/12/12/lex/)

- Bilgisayar bilimlerinde programlama dillerinin tasarımı ve geliştirilmesi sırasında kullanılan ve dildeki kelimelerin analizine ***(lexical analysis)*** yarayan kod üretme programıdır. Yani lex için hazırlanmış bir dosyayı lex programından geçirdikten sonra size C dilinde bir kod çıkar. Bu kodu C dilinde derledikten (compile) sonra çalışan bir programınız olur. Veya tercihen bu çıktıyı ***"yacc"*** programına alt yapı oluşturmak için de kullanabilirsiniz.
- LEX programının ismi inigilizcedeki lexical analyzer kelimesinden gelir. Kelime bilimi anlamına gelen Lexical kelimesinin ilk 3 harfinden kısaltılmıştır. LEX programının linux üzerinde çalışan ve yaygın bir sürümü ***flex*** ismindedir. flex programı da lex gibi çalışmaktadır ve hemen hemen aynı parametre ve özelliklerle kullanılabilir.
- LEX dosyalarının 3 ana bölümü bulunur. İlk bölümde fonksiyon ve değişken tanımlamaları ve projeye dahil edilecek (include) kütüphaneler tanımlanır.
- ikinci bölümde LEX dosyamızın omurgasını oluşturan düzenli deyimler (regular expression) kısmı yer alır. Burada her ihtimal için ayrı bir regular expression tanımlanarak ilgili regular expression'a girilmesi durumunda ne yapılacağı kodlanır.
- Son bölümde ise fonksiyon içerikleri yer alır.

```
%{ // yukarıda anlatılan ilk bölüm tanımlamalar yapılıyor
    #include "y.tab.h"
    #include <stdlib.h>
    void yyerror(char *);
%}
```

```
// ikinci bölüm regular expressionlar
%%
[0-9]+        {
                  yylval = atoi(yytext);
                  return INTEGER;
              }
[-+n]         { return *yytext; }
[ t]          ;        /* skip whitespace */
.             yyerror("Unknown character");
%%
```

```
%%
// son bölüm fonksiyon içerikleri
int yywrap(void) {
    return 1;
}
```

```
Ex1.l :
%%
zippy printf("I RECOGNIZED ZIPPY");

$cat test1
zippy
ali zip
veli and zippy here
zipzippy
ZIP

$cat test1 | ex1
I RECOGNIZED ZIPPY
ali zip
veli and I RECOGNIZED ZIPPY here
zipI RECOGNIZED ZIPPY
ZIP
```

```
%%
zip printf("ZIP");
zippy printf("ZIPPY");

$cat test1 | ex2
ZIPPY
ali ZIP
veli and ZIPPY here
ZIPZIPPY
```

```
%%
monday|tuesday|wednesday|thursday|friday|
saturday|sunday printf("<%s is a day.>",
  yytext);
$cat test3
today is wednesday september 27

$ex3 < test3
today is <wednesday is a day> september 27
```

!!!!!!!!!! Önemli Not: Lex spesifikasyon dosyasının sonunda ekstra boşluk ve / veya boş satır bırakmayın !!!!!!!!!!

❖ **Design Patterns:**

[abc] matches **a, b or c**
[a-f] matches **a, b, c, d, e, or f**
[0-9] matches any digit
X+ matches one or more of **X**
X* matches zero or more of **X**
[0-9]+ matches any integer
(...) grouping an expression into a single unit
(a|b|c)* is equivalent to **[a-c]***
X? X is optional (0 or 1 occurrence)
if(def)? matches **if** or **ifdef** (equivalent to **if|ifdef**)

[A-Za-z] matches any alphabetical character
. matches any character except newline character
\. matches the **.** character
\n matches the newline character
\t matches the tab character
**** matches the **** character
[\t] matches either a space or tab character
[^a-d] matches any character other than **a,b,c** and **d**

Real numbers
[0-9]*(\.)?[0-9]+
To include an optional preceding sign:
[+-]?[0-9]*(\.)?[0-9]+
Integer or floating point number
[0-9]+(\.[0-9]+)?
Integer, floating point or scientific notation.
[+-]?[0-9]+(\.[0-9]+)?([eE][+-]?[0-9]+)?

-**Lex** builds the **yylex()** function that is called, and will do all of the work for you.
-Lex also provides a count **yyleng** of the number of characters matched.
-**yywrap** is called whenever lex reaches an end-of-file
-**Lex** is a tool for writing lexical analyzers.
-**Yacc** is a tool for constructing parsers.

```
/* rule-order.l */
%%
for printf("FOR");
[a-z]+ printf("IDENTIFIER");

for input
for count := 1 to 10
the output would be
FOR IDENTIFIER := 1 IDENTIFIER 10
```

<<<<<<<<<

!!!!!!!!!! Eğer bir ifade belirlenmiş 2 kurala da uyuyorsa ilk önce hangi kurala girdiyse o uygulanır !!!!!!!!!!

<<<<<<<<<

3) *DERS 04 - YACC:* (http://bilgisayarkavramlari.sadievrenseker.com/2008/12/12/yacc/) || (https://www.youtube.com/watch?v=__-wUHG2rfM)

- YACC, bilgisayar bilimlerinin önemli dallarından birisi olan dil tasarımı ve dil geliştirilmesi sırasında (compiler teory) sıkça kullanılan bir kod üretici programdır. YACC basitçe dildeki sözdizim (syntax) tasarımı için kullanılır ve tasarladığımız dildeki kelimelerin sıralamasının istediğimiz şekilde girilip girilmediğini kontrol eder. Aynı zamanda sıralamadaki her kelimenin anlamını da yacc marifetiyle belirleyebiliriz.
- YACC temel olarak BNF (Backus Normal Form) kullanarak cümle dizimini belirtmektedir.
- LEX ile birlikte kullanıldığıdan bir dil tasarımının neredeyse yarısı olan lexical (kelime) ve syntax (cümle) analizi tamamlanmış olur. Bundan sonra dildeki her kelime ve cümle diziliminin anlamını (semantic) kodlamak kalır.
- YACC'i her ne kadar anlatmaya çalışsam da örnekler üzerinden ya da video izlenerek öğrenilmesi daha kolay duruyor. O yüzden bir konunun daha sonuna gelmiş bulunmaktayız.
- Sıkı Çalışın 100 Alın :D

YACC TOKEN'LARI VE ÖZELLİKLERİNİ TANIMLAMA			
%token	Token adlarının belirtir. (Örneğin; %token INTEGER)	%union	Semantic değerler için birden fazla veri türü belirleme.
%left	Sol ilişkilendirmeli operatörleri belirtir.	%start	Start sembolünü bildirir. (Varsayılan kurallar içindeki ilk değişken)
%right	Sağ ilişkilendirmeli operatörleri belirtir.	%prec	Bir kuralın önceliğini atar.
%nonassoc	Kendileriyle ilişkilendirilmeyen operatörleri tanımlar.	$$	Bu devamlıdan (nonterminal) dönecek olan değerdir. (Yani soldaki ifade)
%type	Değişkenlerin türünü bildirir.	$1 / $3	BNF yapısındaki ilk parametredir. / BNF yapısındaki 3. parametredir. Örneğin; | expr '+' expr { $$ = $1 + $3; } => $1 = "1.expr", $2 = "+", $3 = "2.expr"

Cheat Sheet => https://ufile.io/xaa2r

Ek Bilgi => http://comp.eng.ankara.edu.tr/lisans-egitimi/ders-sayfalari/ikinci-sinif-guz-donemi/com241-programlama-dilleri-kavramlari/

Örnekler => https://www.ibm.com/support/knowledgecenter/en/ssw_aix_72/com.ibm.aix.genprogc/ie_prog_4lex_yacc.htm

https://www.epaperpress.com/lexandyacc/prl.html