YOZGAT BOZOK ÜNİVERSİTESİ

# Programlama Dilleri Prensipleri

Dr. Öğr. Üyesi R. Sinan ARSLAN

# Giriş

- Derleme Süreci
- Sözcüksel ve Sözdizim Analizi
- Kod Üretimi
- Derleyici ve Yorumlayıcı Karşılaştırması

# Dillerin Çevrimi

- Yüksek düzeyli bir dilde yazılmış bir programın(kaynak kod), yürütülebilmesi için bilgisayarın doğrudan tanıdığı tek bir dil olan makine diline çevrilmesi sürecine derleme denir.

```
int main(int a, int b){
        printf();
        scanf();
        for loop
        if
        …
}
```

Derleme

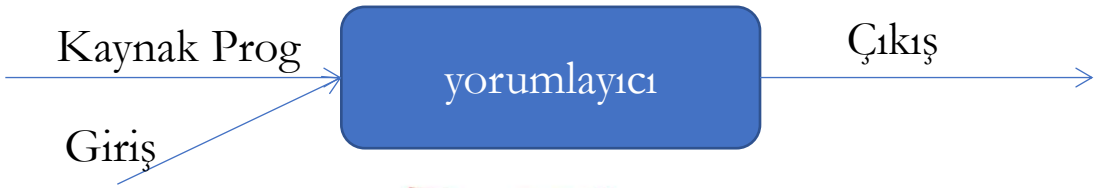
```
0x 60 00 00 80
0x A4 00 00 00
0x 60 01 00 84
0x A4 01 01 00
0x 60 02 00 00
0x 60 03 00 04
0x 60 04 00 00
0x 60 05 00 01
0x 08 00 00 02
0x 20 00 00 03
0x 20 04 04 05
0x 11 20 04 01
```

# Dillerin Çevrimi

- Bu çevrim işlemi için geliştirilen araçlara derleyici(compiler) denir.
- kaynak kodun makine diline çevrilmiş biçimine hedef kod(object code) denir.
- Makine dilindeki karşılığı oluşturulan programa girişler alınarak yürütme(implementation) gerçekleşir.

# Dillerin Çevrimi

- Derleme işleminde kaynak programın tamamı makine diline dönüştürülür
- Makine diline dönüştürülmüş kod yürütülür.
- Makine koduna dönüşüm derleyici tarafından yapılır.
- Bazı diller derleyici yerine yorumlayıcı da kullanabilir.
- Bir yorumlayıcı, bir programın her satırını birer birer makine diline çevirir ve o satırla ilgili bir alt programı çağırarak o satırın çalıştırılmasının sağlar. Satır satır çevrim yapılır.

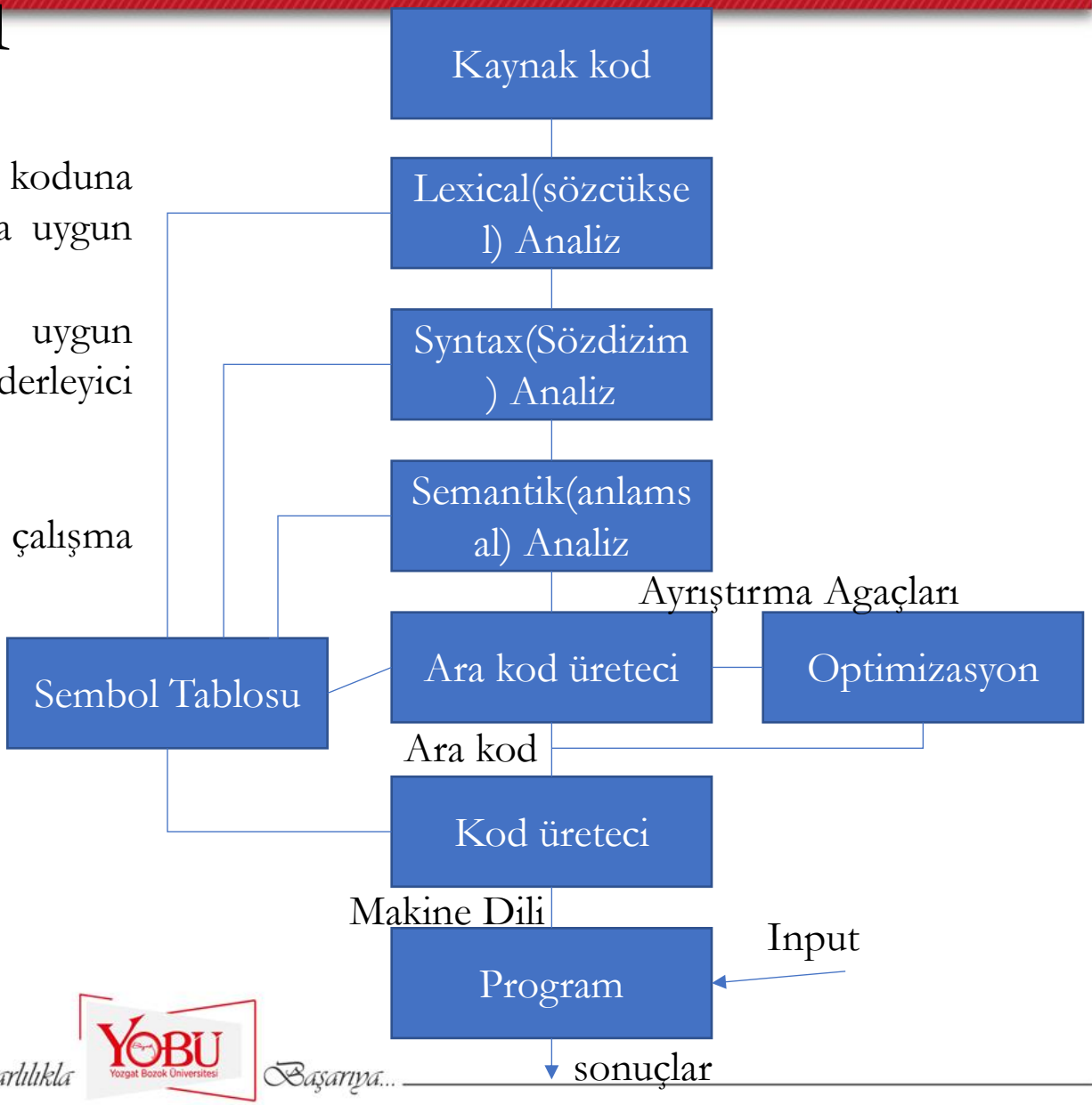


# Derleme Süreci

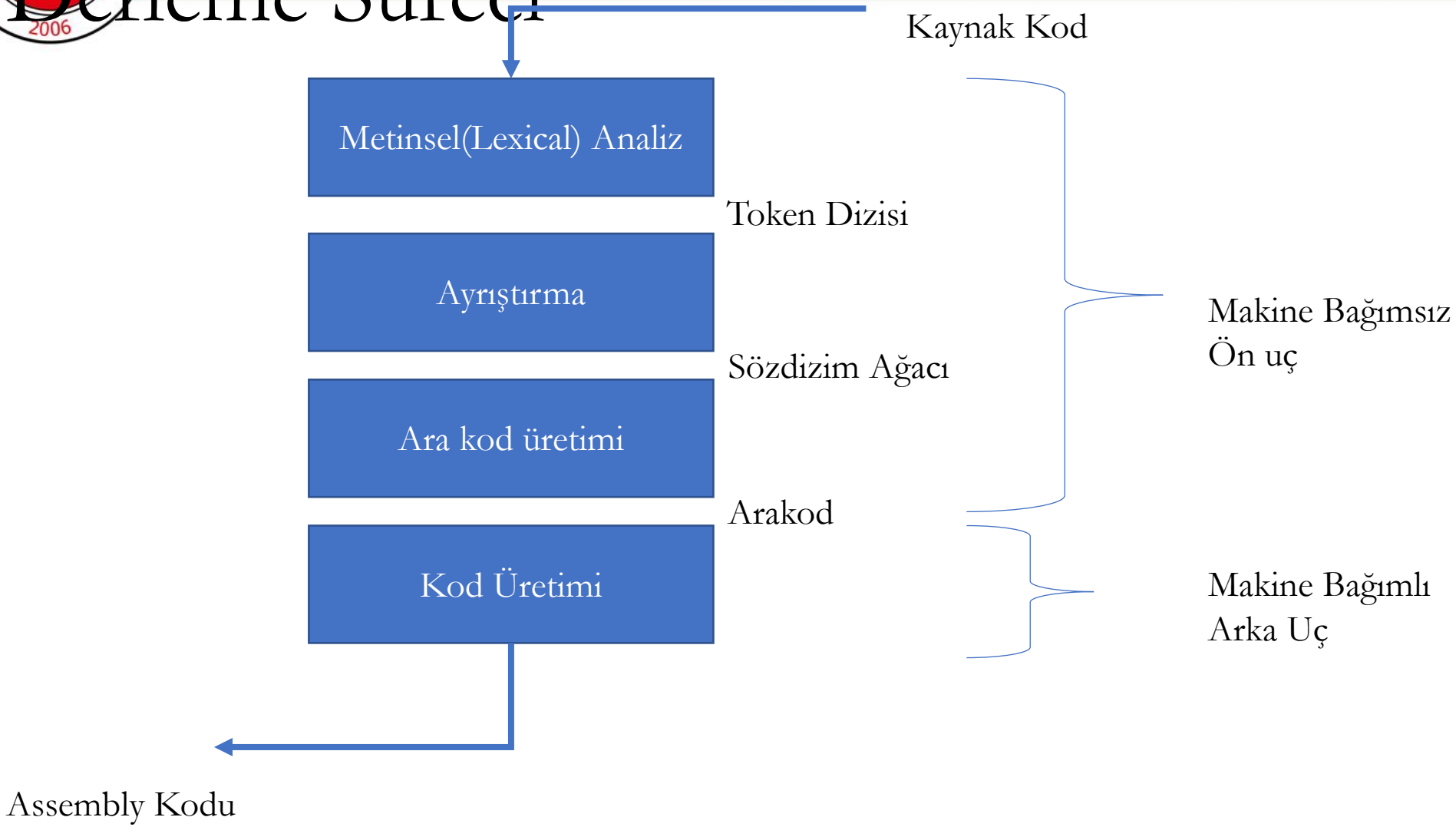
- Bir derleyici kaynak kodu hedef koda dönüştürürken öncelikle baştan sonra tüm programın kaynak kodunun o dilin kurallarına uygun oluşturulup oluşturulmadığını inceler.
- Program metnini oluşturan bütün simgelerin o dilin alfabesinde bulunup bulunmadığını, bu simgelerin kurallara uygun oluşturulup oluşturulmadığı ve uygun bir deyim listesi olup olmadığı kontrol edilir.
- her isim ve değişkenin bir işlemin sonucunu doğru üretecek şekilde tanımlandığını yani anlamlılığı inceler.
- Derleyicinin çalışması derleme zamanı (compile time) ve çalışma zamanı(runtime)olmak üzere iki aşamada tamamlanabilir.

# Derleme Süreci

Örnek bir kaynak kod makine koduna dönüştürülürken, öncelikle kodun kurallara uygun olup olmadığı kontrol edilir.
Dilin alfabesine uyumluluk, kurallara uygun oluşturulma, uygun deyim listesi oluşturma derleyici tarafından kontrol edilir.
Sonrasında metnin anlamlılığı incelenir.
Derleyici derleme zamanında ve ya çalışma zamanında çalışabilir.
Derleme süreci solda gösterilmiştir.



# Derleme Süreci



# Sözcüksel(lexical) analiz

- Kaynak programın en alt düzeyli birimlerini(lexem) belirler ve ayırt eder. tek başına bir anlam taşıyan karakter katarlarını tanır ve bunları ayırır.
- Dile ait anahtar sözcükleri(if, while vs.), değişken adlarını (i,j,ad, soyad), sabitler(5.45), noktalama işaretlerini ve operatörleri ayırt eder.
- Bunları sınıflayarak bir token dizisi oluşturur.
- Boşluklar, açıklamalar gibi bölümleri siler.

# Sözcüksel Analiz

```
program writeprogram(input, output);
var b, z: integer;
begin
read(b,z);
while b<>z do
if  b>z then b := b-z else z: b-z;
write (b )
end
```

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| program | writeprogram | ( | input | , | output | ) | ; | |
| var | b | , | z | : | integer | ; | begin | |
| read | ( | b | , | z | ) | ; | while | |
| b | < | > | z | do | if | b | > | |
| z | then | b | : | = | b | - | z | |
| else | z | : | b | - | z | ; | write | |
| ( | b | ) | end | | | | | |

token list

# Sözdizim(Syntax) Analizi

- Kaynak programı oluşturan sözcüklerin programlama dilinin gramer kurallarına uygun bir **sıralamada** olup olmadıklarını belirlemek için sözdizim analizi yapılır.
- Bunun için BNF ve CFG yapılarından yararlanılır.
- Söz dizim analizi lexical analiz aşamasının oluşturduğu token dizisini giriş olarak alır ve gramer aracının kullanarak bir parse ağacı oluşturur.
- Eğer bu token dizisi için gramer bir parse ağacı oluşturuyorsa bu ifade söz dizim kuralları yönüyle doğru yazılmıştır ve gramer bu katarı üreten türetim adımlarını verir.
- Aksi durumda hata mesajları üretilir.

# Sözdizim Analizi

ARİTMETİK İŞLEMLER İÇİN BİR GRAMER

İfade -> İfade ekleme_operatörü terim | terim

terim -> faktör | terim çarpma_operatörü faktör

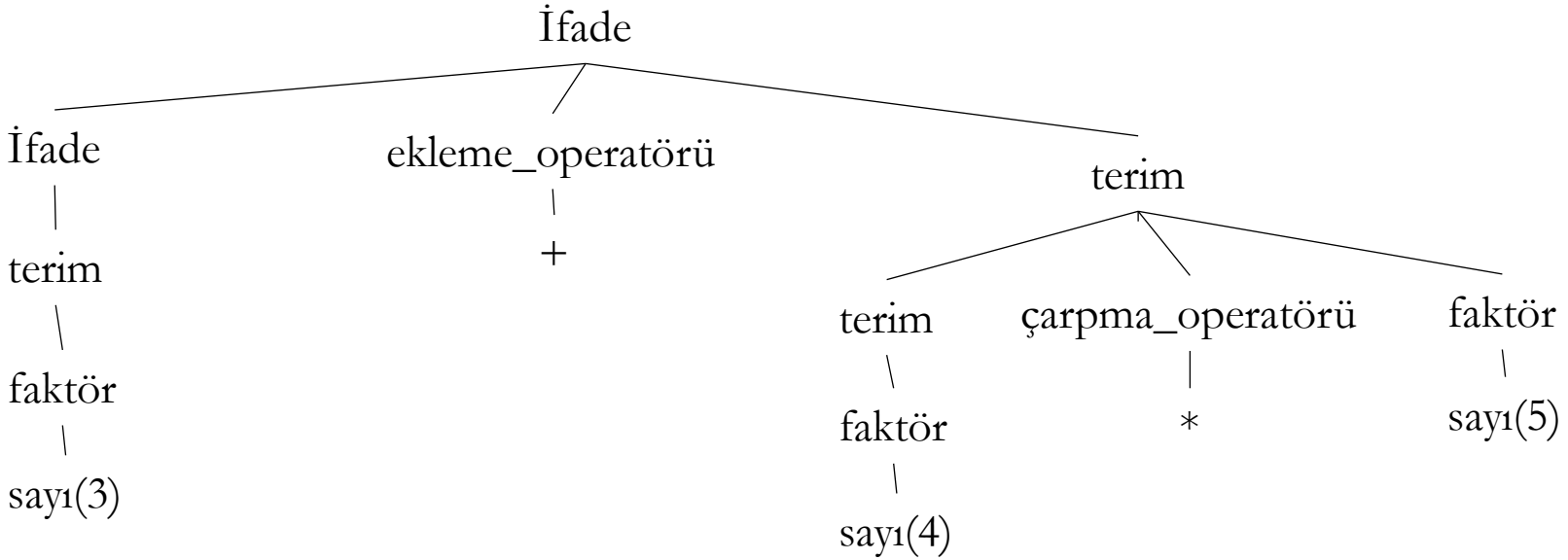
faktör -> sayı | tanımlayıcı | -faktör | (İfade)

ekleme_operatörü -> + | -

Çarpma_operatörü-> +| /

3+4*5 ifadesinin gramer ağacına göre kontrolünü yapınız.

# 3+4*5 ifadesi ayrıştırma ağacı



# 3+4*5 ifadesi türetim adımları

İfade ekleme_operatörü terim
terim ekleme_operatörü terim
faktör ekleme_operatörü terim
sayı ekleme_operatörü terim
3 ekleme_operatörü terim
3 + terim
3 + terim çarpma_operatörü faktör
3 + faktör çarpma_operatörü faktör
3 + sayı çarpma_operatörü faktör
3 + 4 çarpma_operatörü faktör
3 + 4 * faktör
3 + 4 * sayı
3 + 4 * 5

Bu türetim adımları oluşturulurken art arda gelen türetimlerde hep en soldaki nonterminalden türetim gerçekleştirilmiştir. Parser olarak ifade edilen sözdizimsel analiz yapan ve derleyicinin içinde bulunan altprogram bütün bir kaynak kodu bazı hatalar bulunsa bile ayrıştırması beklenir. Sözdizimsel analizde hata iyileştirme stratejileri kullanılır.

# Anlamsal(Semantic) Analiz

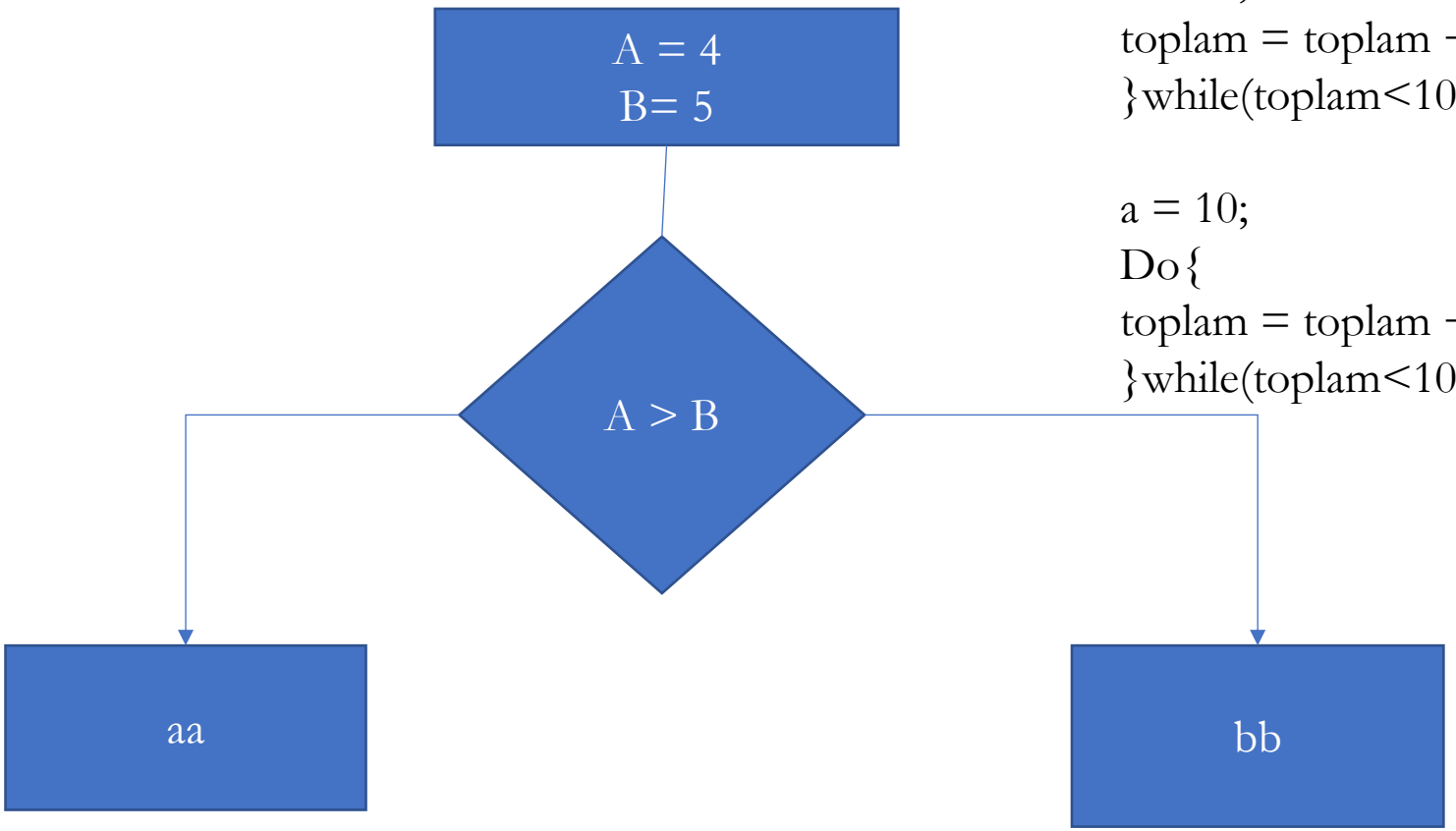
- Anlam çözümleme, kaynak program için sözdizim çözümleme sırasında oluşturulmuş ayrıştırma ağacı kullanılarak, soyut bir programlama dilinde bir program oluşturulmasıdır.
- Grammer sadece dilin sözdizimini tanımlayabilir.
- Dilin çevrilmesi sırasında gerek duyulan bazı bilgileri tanımlamada regüler ifade yada grammer kullanılmaz.
- Bu özelliklere anlamsal özellikler(semantic features) denir.
- Bunlar anlamsal kurallar ile belirlenir.
- Ayrıştırma ağacındaki her düğüm bir grammer simgesine karşı düşer.
- Grammer simgeleri kendileriyle ilişkilendirilen niteliklere sahip olabilirler.
  - Tip, adres, sayı, işaretçi gibi.
- Her nitelik yapısı bir parse ağacı düğümünün alt alanı olacaktır.
- Her grammer kuralının kendisiyle ilişkilendirilmiş olan bir anlamsal kurallar kümesi bulunur.
- Anlamsal kurallar, o grammer kuralında yer alan simgelere ait niteliklerin ne şekilde hesaplanacağını belirler ve diğer işlemleri yönelendirirler. (bilgi girişi, ara kod üretimi vb.)
- Anlamsal çözümleme sonucunda üretilen kod ara diller(soyut), kaynak dilin veri türleri ve işlemleriyle uyumlu olarak tasarlanmış, hayali bir makine dili olup, derleyicinin kaynak ve amaç dilleri arasında bir ara adım oluşturur.

# Kod optimizasyonu

- Üretilen kodun, olası çalışma zamanını yada bu kodu saklamak için gerekli bellek alanını azaltmak için yapılacak iyileştirmeler sürecidir.
- Bu iyileştirmeler parser çıkışı ara kod üzerinde global yapılabileceği gibi kod üretildikten sonra üretilen kod üzerinde localde yapılabilir.
- Kod daha etkin yapmak için
  - Kod parçasında ortak alt ifadelerin kaldırılması,
  - Ölü kodun kaldırılması
  - Çevrim sabitinin çevrim dışına aktarılması gibi iyileştirmeler yapılmalıdır.

https://www.geeksforgeeks.org/code-optimization-in-compiler-design/

# Kod optimizasyonu




Ölü kod

```
Do{
a = 10;
toplam = toplam + a;
}while(toplam<100);

a = 10;
Do{
toplam = toplam + a;
}while(toplam<100);
```

# Ara Kod üretimi

- Ara koddan hedef donanıma uygun makine koduna dönüşüm yapılmalıdır.
- Three address en bilinen ara kod üretim tekniğidir.
  - A = b * c + d

    t1 = b * c

    t2 = t1 + d

    A = t2
- Ara kod üretimi sayesinde farklı diller için farklı makine kodu üretiminde avantaj sağlanır.

# Ara kod üretimi örnek

```
İnt main()
        int dizi1[3], dizi2[4],
carpim i;
        int *p;
        int *q;
        carpim = 0;
        p = dizi1;
        q = dizi2;
        for(i 1 to 10)
                carpim +=
*p++ * *q++;
        return 0
```

```
Carpim = 0;
p = &dizi1;
q  = &dizi2;
i  =0;
S2: if(i >= 10) goto S3;
        t1 = *p;
        t2  = p + 1
        p = t2;

        t3 = *q;
        t4 = q+1;
        q = t4;

        t5 = t1 * t3;
        t6 = carpim + t5;
        carpim = t6;
        t7 = i + 1
        i = t7;
Goto S2;
S3:
```

# Ara kod üretiminin avantajları

- Örnek olarak Java, C , C++, C# dilleri için 5 farklı makine koduna dönüşüm gerektğinde toplam 20 adet optimizasyon ve kod üretim yapılmalıdır.
- Çünkü daha önce de söylediğimiz gibi optimzasyon ve kod üretimi her bir makine için ayrı ayrı yapılmalıdır.
- Derleyici tasarımında en zor kısım kaynak kodun makine koduna dönüşüm aşamasıdır.
- Bu sebeple 4 farklı dil için derleme yapıldığında ara kod üretimi olmaz ise, 4 ön uç, 4x3 =12 optimize edici işlem, 12 kod üretici gerekli iken ara kod kullanıldığında, 4 ön uç, 1 optimize edici 3 kod üretici yeterli olacaktır.
- Ön uç makine değişiminde hep sabit kalır. Arka uç değişikliğe uğrar.

# Derleyici vs Yorumlayıcı

- Yorumlayıcı diller de kod satır satır çalıştırılır. Bu sebeple çevrim her bir satır için tekrarlanır. Lisp dili yorumlayıcı dile örnek olarak verilebilir.
- Derleyicide program için gerekli çevrim süreci bir kez yapılır.. Üretilen makine kodu tekrar tekrar kullanılır. Programı tümü için makine kodu tek seferde oluşturulur.
- Derleme yorumlayıcıya göre zamandan kazanç sağlar.
- Yorumlayıcı derleyiciye göre daha az bellek alanı kullanır.
- Yorumlayıcı da kodda bulunan hatalar adım adım programcıya gösterilirken, derleyici de tüm kod makine koduna dönüştürüldükten sonra bulunan tüm hatalar toplu bir şekilde programcıya gösterilir.

# Derleyici vs Yorumlayıcı

- Java dilinde Java byte kod oluşturulur ve bu kod yorumlanır. Hem derleme hem yorumlama yapılır.
- Perl dilinde hata tespiti için önce derleme sonra yorumlama gerçekleştirilir.

# İki farklı dilin bir arada kullanılması

- Gerekli hallerde kodun yazımı sürecinde farklı dillerin bir arada çalıştırılması istenebilir.
- Bu durumda tüm kodun hedef dile çevrimi sonrasında eksik olan kod bölümünün tamamlanması yapılabilir. Ancak bu zor ve zahmetli olacaktır.
- Bunun yerine gerekli hallerde farklı diller arasında geçici olarak o dili çağırarak kullanmakta mümkündür.
- Örneğin java kodu içerisinde herhangi bir hesap yapan Python kodunu çağırıp kullanabilmek mümkündür.

# Java-Python birlikte kullanım örneği

```
public static void main(String[] args) throws ScriptException, IOException {

    StringWriter writer = new StringWriter(); //ouput will be stored here

    ScriptEngineManager manager = new ScriptEngineManager();
    ScriptContext context = new SimpleScriptContext();

    context.setWriter(writer); //configures output redirection
    ScriptEngine engine = manager.getEngineByName("python");
    engine.eval(new FileReader("numbers.py"), context);
    System.out.println(writer.toString());
}
```

https://bytes.com/topic/python/insights/949995-three-ways-run-python-programs-java

# Konu ile ilgili web siteleri

https://medium.com/@abdulsamet.ileri/compilers-ve-interpreters-eacd14a227c4

https://www.geeksforgeeks.org/three-address-code-compiler/

# Sorular

- Yorumlayıcının derleyiciye göre avantajları nelerdir? Araştırınız.
- Üçlü adres kodu ne demektir? En az 50 satırlık bir C kodu yazarak bunu üçlü adres kodu olarak ifade ediniz.
- Örnek bir BNF grammeri yazınız. Bu gramere 1 adet uygun ve 1 adet uygun olmayan ifadeyi ayrıştırma ağacı çizerek gösteriniz.
- Verilen bir dizinin elemanlarının toplamını bulan bir c# kodu yazınız. Bu kodu hem Python, hem Java hemde bir C kodu içerisinden çağırarak çalıştırınız.