Programlama Dillerinin Prensipleri

HAFTA 3
DİLLERİN ÇEVRİMİ VE TASARIMI

İçindekiler

- Dillerin çevrimi
- Derleme süreci
- Sözcüksel (lexical) analiz
- Tokenlar ve heceler
- Sözdizim analizi
- İfade notasyonu
- Dilbilgisi (Gramerler)
- Anlamsal (semantic)analiz
- Kod optimizasyonu
- Kod üretimi
- Derleyici ve yorumlayıcı karşılaştırılması

Dillerin çevrimi

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

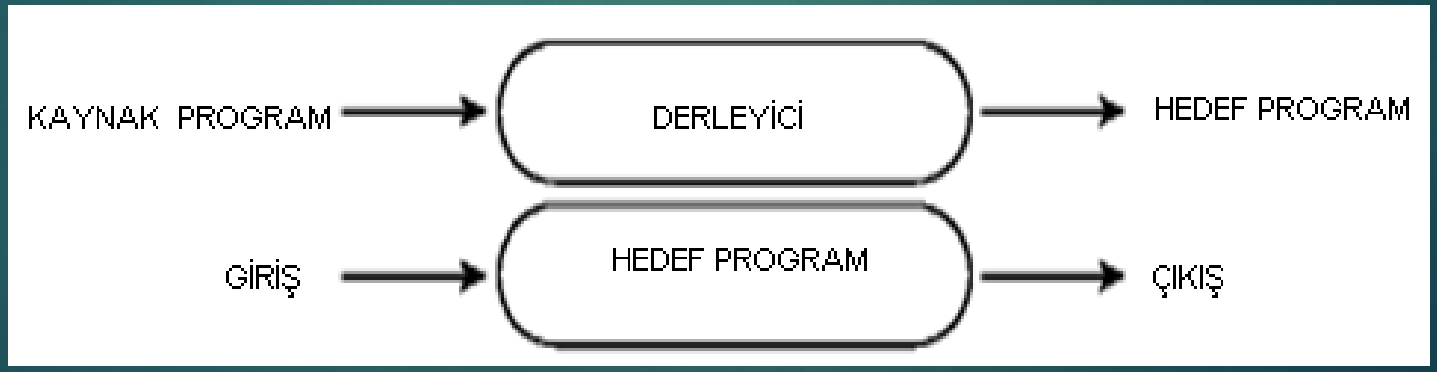
```
program cevir(input, output);
var i, j: integer;
begin
      read(i, j);
      while i <> j do
            if i > j then i := i - j;
            else j := j - i;
      writeln(i)
end.
```

Derleme

```
27bdffd0 afbf0014 0c1002a8 00000000 0c1002a8 afa2001c 8fa4001c
00401825 10820008 0064082a 10200003 00000000 10000002 00832023
00641823 1483fffa 0064082a 0c1002b2 00000000 8fbf0014 27bd0020
03e00008 00001025
```

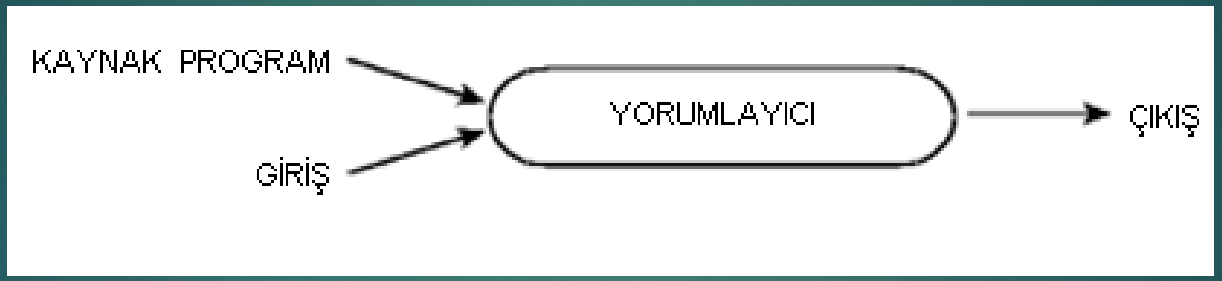
Dillerin çevrimi

- 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 ve yürütülür. Derleyicinin diğer bir alternatifi olan dil çeviricisi yorumlayıcıdır.



- Bir **yorumlayıcı**, bir programın her satırını birer birer makine diline çevirir ve o satırla ilgili bir altprogram çağırarak o satırın çalıştırılmasını sağlar.

Derleme Süreci

- Bir derleyici kaynak kodu hedef koda dönüştürürken, öncelikle bu kaynak kodun kurallara uygun oluşturulup oluşturulmadığını inceler.
- Program metnindeki her isim ve değişkenin bir işlemin sonucunu doğru üretecek biçimde tanımlanmışlığı da yani program metninin anlamlılığı da incelenir.

Derleme Süreci

- Derleyicinin çalışması **derleme zamanı** (*compile time*) ve **çalışma zamanı** (*run time*) olmak üzere iki aşamada tanımlanabilir.
- Derleme sırasında geçen zamana **derleme zamanı** ve programların çalışması sırasında geçen zamana da **çalışma zamanı** denir.
- Çalışma zamanı derleme sürecinden bağımsızdır.

Derleme Süreci

- Derleme süreci 2 kısımdan oluşur.
  - Birinci kısım, bir programın sözdizimini ve anlamını dilin kurallarına göre inceleyerek çözümleyen ve bir ara kod oluşturan **ön-uçtur.**
  - İkinci kısım ise programın ara kod gösteriminde isteğe bağlı eniyileme uygulayarak makine kodunu oluşturan **arka-uç** (*back-end*).

Derleme süreci

- Derleyicinin ön ucuna analiz aşaması denilir ve bu kısım makinden bağımsız olarak gerçeklenir.
  - Bu süreçte, metinsel çözümleme sonucunda token dizisi, sözdizim çözümlemesinin sonucunda ayrıştırma ağacı ve anlam çözümlemenin sonucunda da bir ara kod ile ifade edilen soyut program oluşturulur.
- Derleyicinin arka ucunda gerçeklenen kısımlar sentez aşamasıdır.
  - Burada ara kod kımında bazı optimizasyon uygulamaları yapılarak makine kodunun hızlı çalışması için tedbirler alınabilir.
  - İsteğe bağlı bir aşamadır. Kod üretimi kısmı ise makine kodunun üretildiği kısımdır.

Sözcüksel (lexical) analiz

- Kaynak programın en alt düzeyli birimlerini (lexeme) belirler ve ayırdeder.
- Tek başına bir anlam taşıyan karakter katarlarını tanır ve bunları ayırır.
- Dile ait anahtar sözcükleri (for, if, while, vs), değişken adlarını (“i”, “j”, “sayac”), sabitleri (3.14159, 17) ve “(“, “)”, “,”, “+” gibi noktalama İşaretleri ve operatörleri tesbit eder ve bunları sınıflayarak bir token dizisi oluşturur.

Sözcüksel (lexical) analiz

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

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

- Tarayıcının programın tokenlarını oluşturulması

Tokenlar ve heceler

- Bir programlama dilinin söz dizimi token yada terminal adı verilen birimlerle belirlenir.
- Metinsel söz dizim, yazılan program metni ile gramerdeki tokenların diziliminin örtüşmesini açıkça ifade eder.
- Dil içerisinde birim olarak kullanılan alfabetik karakter serilerine **anahtar kelimeler** denir.
- Örnek olarak **if, for** ve **while** Pascal ve C'nin her ikisinde de anahtar kelime olarak kullanılır. Anahtar kelimeler ayrılış kelimelerdir (reserved words) ve değişken ismi olarak kullanılmazlar.

Tokenlar ve heceler

İkili İşlem	Sembol	Pascal	C/C++/java	Lisp
Küçük	<	<	<	<
Küçük–Eşit	≤	<=	<=	<=
Eşit	=	=	= =	==
Eşit değil	≠	<>	! =	/=
Büyük	>	>	>	>
BüyükEşit	≥	>=	> =	>=
Toplama	+	+	+	+
Çıkarma	-	-	-	-
Çarpma	*	*	*	*
Bölme,reel sayı	/	/	/	/
Bölme,tam sayı	**div**	div	/	/
Moduler	**mod**	mod	%	mod

Tokenlar ve heceler

- Bir programlama dilindeki en düşük düzeyli sözdizimsel birimlere **lexeme** adı verilir.
- Programlar, karakterler yerine lexeme'ler dizisi olarak düşünülebilir.
- Bir dildeki lexeme'lerin gruplanması ile dile ait **token'**lar oluşturulur.
- Programlama dilinin metinsel sözdizimi, token'lar ile tanımlanır.
  - Örneğin bir tanımlayıcı (identifier); "ortalama" veya "kök" gibi lexeme'leri olabilen bir token'dır.

Tokenlar ve heceler

- Bazı durumlarda, bir token'ın sadece tek bir olası lexeme'i vardır.
- Örneğin, çıkarma operatörü denilen aritmetik işlemci "-" sembolü için, tek bir olası lexeme vardır. Boşluk (space), ara (tab) veya yeni satır karakterleri, token'lar arasına yerleştirildiğinde bir programın anlamı değişmez.
- Aşağıda b * b – 4 * a * c ifadesinin token dizisi elde edilmiştir.
- İsimler için **name** ve tamsayılar için **number** kullanılırsa, token dizisi

$$\mathbf{name}_b * \mathbf{name}_b - \mathbf{number}_4 * \mathbf{name}_a * \mathbf{name}_c$$

gösterimiyle temsil edilir.

Sözdizim analizi

- Kaynak programı oluşturan sözcüklerin programlama dilinin gramer kurallarına uygun bir sıralamada olup olmadıklarını belirlemek için söz dizim analizi yapılır.
- Bunun için dilin gramer kurallarını ifade eden BNF ve CFG gibi araçlardan yararlanılır.
- Söz dizim analizi lexical analiz aşamasının oluşturduğu token dizisini giriş olarak alır ve gramer aracını kullanarak bir parse ağacı oluşturur.
- Eğer bu token dizisi için gramer bir parse ağacı oluşturuyorsa bu ifade söz dizim kuralları yönüyle doğru yazılmıştır. Aksi durumda, hata mesajları (sentaks hataları) üretir.

Sözdizim analizi

- Aşağıda aritmetik ifadeler için bir gramer tasarlanmıştır.

ifade→ ifade ekleme-operatörü terim | terim

terim →faktör | terim çarpma-operatörü faktör

Faktör→sayı| tanımlayıcı | - faktör | (ifade)

Ekleme-operatörü→+|-

Çarpma-operatörü→*|/

İfade notasyonu

- Programlama dilerinde a + b * c gibi ifadelerin kullanılması yüksek düzeyli dillerin tasarımında bir başlangıç noktasıdır.
- Örneğin FORTRAN dilinde

(-b + $\sqrt{b^2 - 4*a*c}$) / (2*a)

ifadesi kolaylıkla

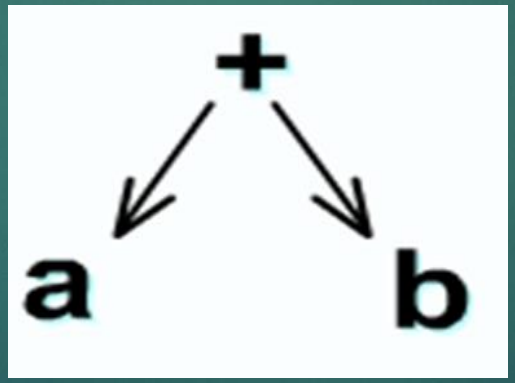
(- b + sqrt(b*b – 4.0*a*c)) / (2.0*a)

biçiminde yazılabilmektedir.

- Infix notasyonunda, işlemci a + b ifadesinde olduğu gibi işlemlerin arasına yazılır. Bu ifade prefix notasyonunda + a b, postfix notasyonunda a b + dizilimindedir.

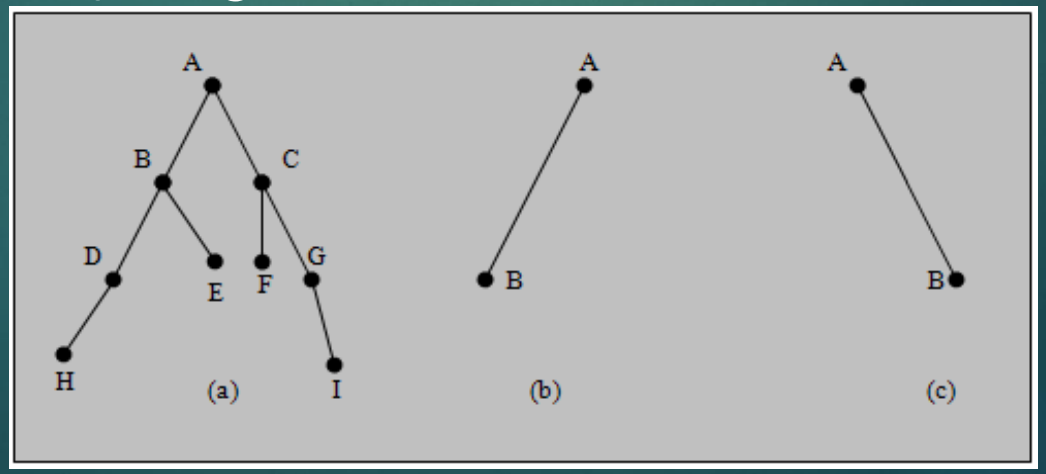
Soyut sözdizim ağaçları

- Bir dilin soyut söz dizimi, dildeki her yapının anlamlı bir açıklamasıdır.
- + a b Prefix ifadesi, a+ b infix ifadesi ve a b + postfix ifadelerinin hepsi gösterimleri farklı da olsa aynı anlamlıdır.

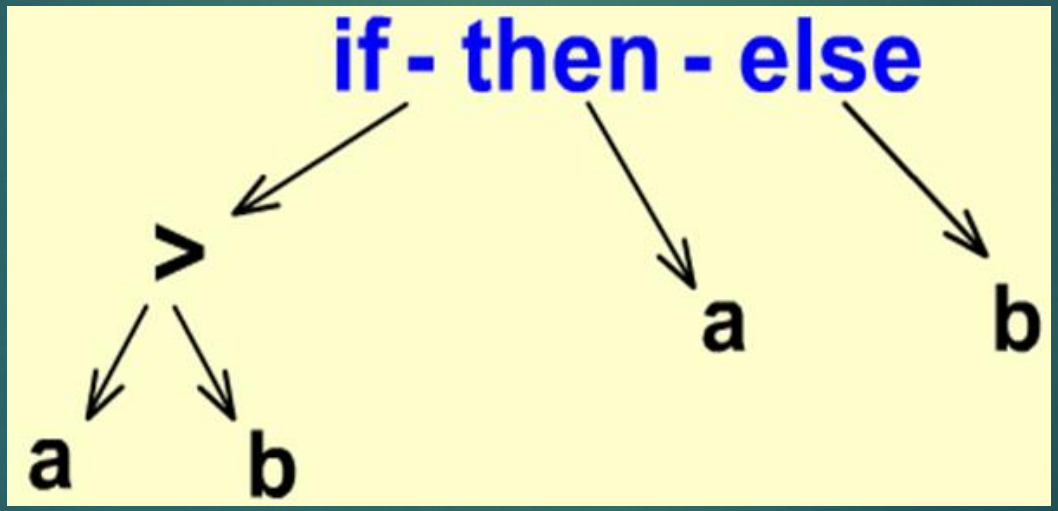


İfadelerin ağaç ile gösterimi

- A-B gibi bir aritmetik ifade de A ve B nin sırası önemlidir.
- Bir ikili ağaçta (binary tree) her bir düğümün sadece sol çocuk ve sağ çocuk olmak üzere iki tane çocuğu vardır.
- Buna göre bir ikili ağacın her düğümünün 0,1 ya da 2 tane çocuğu vardır.



Soyut söz dizim ağaçları, tümce yapıları için uygun operatörler kullanılarak diğer yapılara genişletilebilir. Bu soyut ifadenin ağacı **if** – **then** – **else** operatörü üretilerek çizilebilir.



Dilbilgisi (Gramerler)

- **Gramer**, bir programlama dilinin metinsel (somut) sözdizimini açıklamak için kullanılan bir gösterimdir.
- Gramerler, anahtar kelimelerin ve noktalama işaretlerinin yerleri gibi metinsel ayrıntılar da dahil olmak üzere, bir dizi kuraldan oluşur.
- Dilin somut söz dizimi, anahtar kelimelerin yeri ve noktalama işaretleri gibi söz dizimsel detayları içeren yazılı ifadeler dilin gramerini oluşturur.
- BNF (Backus – Naur Formu) ve CFG (Context-Free Gramerler) gibi gramerler somut söz dizimini açıklamak için geliştirilmiş gösterim araçlarıdır.

Dilbilgisi (Gramerler)

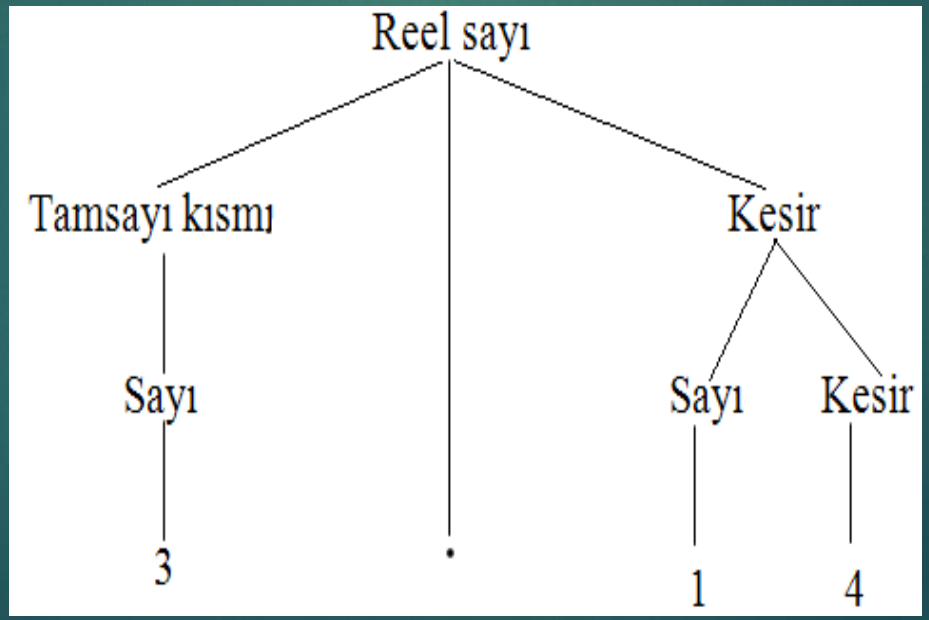
- CFG: İçerikten bağımsız dilbilgisi gösterim şeklidir. Belirsizlik içeren durumlar CFG ile gösterilemez. Anlamlı ve belli olan kurallar gösterilebilir.

- BNF: CFG gramerlerini matematiksel ve formal yolla daha kısa ve kolay bir şekilde açıklamayı amaçlar.

- EBNF (Extended BNF): BNF gösteriminin genişletilmiş halidir. EBNF'te Seçimlik (optionality), Yineleme (repetition) ve Değiştirme (alternation) olmak üzere üç özellik yer almaktadır. EBNF'de okunabilirlik ve yazılabilirlik daha iyidir ve EBNF'nin avantajı gösterimi kolaylaştırmasıdır. Burada dilde yenilik yoktur. EBNF ile ifade edilebilecek herşey BNF ile de ifade edilebilir.

Dilbilgisi (Gramerler)

- Dilin gramer kuralları parse ağacı olarak adlandırılan hiyerarşik bir yapıya sahiptir. Örnek olarak dil içerisindeki reel sayılar için 3.14 serisi aşağıdaki parse ağacında gösterilmiştir:



Dilbilgisi (Gramerler)

- Parse ağacının tepesindeki yapraklar (3, . , 1, 4) "**terminaller**" olarak isimlendirilir.
- Terminallerin alt düğümleri olmaz, uç simgelerdir ve tek başlarına simgelenir.
- Terminaller dilin alfabesindeki simgelerden oluşurlar.
- Parse ağacının diğer düğümleri ise (Reel sayı, sayı, Tamsayı kısmı, Kesir, ve reel sayılar gibi) **nonterminal** olarak isimlendirilir.

Dilbilgisi (Gramerler)

- Nonterminaller dilin sözdizim değişkenleridir ve dilin yapısını dil yapısını simgelerler.
- Her non terminal bir terminale doğru gitmelidir ve böylece parse ağacındaki her düğüm bir sonuca dayanır.
- Yukarıdaki parse ağacının anlamı, Reel bir sayı, Tam sayı, bir nokta, kesirli kısım olarak ifade edilir ve reel bir sayının tamsayıyı izleyen bir nokta ve noktadan sonra kesirli kısım gelmesi gerektiğini ifade eder. Buna gramerin **türetim kuralı** denir.

Dilbilgisi (Gramerler)

- Programlama dili tasarlamak için kullanılan gramer "Başlangıç nonterminali", "nontermineller", "terminaller" ve "kurallar" olmak üzere 4 kısımdan oluşur.
- Terminaller ; dilin alfabesinin simgeleridir ve terminallerden yeni üretimler yapılamaz.
- Nonterminaller; bir programlama dilindeki yapıları gösteren söz dizim değişkenleridir.
- Kurallar kümesi ; bir tümcesel yapının bileşenlerinin saptanması için kullanılır. Her bir terminal bir nonterminalden belirli kurallarla oluşturulur.
- Dördüncü bileşen olan "başlangıç nonterminali" ise türetime başlamak için kullanılır

Dilbilgisi (Gramerler)

- Nonterminalle terminal arasında **“::=”** yada “→”dir kullanılır “olabilir” diye okunur.
- BNF (Backus – Naur Formu) yukarıda bahsedilen bileşenleri kullanarak dil tasarımında kullanılan bir metafildir.
- BNF içerisinde nonterminaller “< >” şeklinde özel semboler içerisine alınır ve boş dizi **<boş>** şeklinde yazılır. + ve * şeklinde semboller içeren terminaller genellikle bu şekildedir.
- Dil kurallarının nonterminalleri <reel sayı>, <tam sayı - kısım>, <kesir> ve <sayı> olarak yazılır. Gösterimler 0,1,….,9 sayıları ve **ondalık** noktadır.

Türetimler

- **: : =** sembolünün okunuşu "olabilir" ve **|** sembolünün okunuşu "veya" gibidir.

<reel sayı> : : = <tam sayı - kısım> . <kesir>

<tam sayı - kısım> : : = <sayı> | <tam sayı - kısım> <sayı>

<kesir> : : = <sayı> | <sayı> <kesir>

<rakam> : : = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<kesir> : : = <rakam> | <rakam> <kesir>

Reel sayılar için BNF kuralları

Parse ağacının somut sentaks tanımlanması

- Her bir yaprak boş katarı gösteren bir terminal veya <boş> ile etiketlenmiştir.
- Her bir yaprak olmayan düğüm bir nonterminal ile etiketlenmiştir
- Bir yaprak olmayan düğümün (nonterminal) etiketi bir ürünün sol tarafıdır ve düğümün çocuklarının etiketleri soldan sağa doğru o ürünün sağ tarafını biçimlendirir.
- Kök düğüm başlangıç nonterminali ile etiketlenmiştir.
- Bir parse ağacı soldan sağa terminalleri okuyarak biçimlendirilen katarı meydana getirir. Bir dildeki katar için bir parse ağacı oluşturulabiliyorsa, bu katar o dile ait bir katardır. Bir parse ağacının oluşturulması işlemine ayrıştırma (parsing) denir.

Dilbilgisi çeşitleri

- Diğer gramer tasarlama aracı olan EBNF (Extended BNF), BNF gösteriminin genişletilmiş halidir.
- EBNF'te Seçimlik (optionality), Yineleme (repetition) ve Değiştirme (alternation) olmak üzere üç özellik yer almaktadır.
- EBNF'de okunabilirlik ve yazılabilirlik daha iyidir ve EBNF'nin avantajı gösterimi kolaylaştırmasıdır.
- Burada dilde yenilik yoktur. EBNF ile ifade edilebilecek herşey BNF ile de ifade edilebilir.

Dilbilgisi çeşitleri

- ***Yineleme :***

BNF'nin genişletilmiş sürümünde süslü parantezler { } belirtilen kataın 0 veya daha fazla tekrarını ifade eder.

- Örneğin

{<ifade>;} yazımı 0 ya da daha fazla **ifade** nonterminalini noktalı virgülle sonlandırarak ardarda getirir.

<ifade_listesi> ::= {<ifade>;}

yazımı BNF'deki şu ifade çiftine denktir:

<ifade_listesi> ::= <boş>

| <ifade> ; <ifade_listesi>

Dilbilgisi çeşitleri

- ***Seçimlik :***

Diğer bir genel eklenti köşeli parantezleri [ ] isteğe bağlı bir yapıyı belirlemek için kullanmaktır.

Örneğin gerçek sayıların isteğe bağlı tam kısmı şu şekilde ifade edilir.

<gerçel sayılar>::=[tam kısım].<kesirli ifade>

Bu ifade BNF'deki şu gösterim çiftine denktir.

<gerçel sayı>::=<tam kısım> . <kesirli ifade> | . <kesirli ifade>

Dilbilgisi çeşitleri

- ***Değiştirme :***

* ve + sembolleri ile daha kısa kurallar ifade edilebilir.

* Simgesinin anlamı sıfır veya dah fazla tekrarlanması;

+ simgesinin anlamı 1 veya daha fazla tekrarlanmasıdır.

Örneğin id tanımlaması yapılırken;

BNF'de: <id> ::= <karakter> | <id><karakter> |<id><rakam>

EBNF'de; <id>::= <karakte (<karakter> | <rakam>)*

Anlamsal (semantic) analiz

- **Anlamsal çözümleme**, kaynak program için sözdizim çözümleme sırasında oluşturulmuş ayrıştırma ağacı kullanılarak, soyut bir programlama dilinde bir program oluşturulmasıdır.
- Gramer sadece dilin sözdizimini tanımlayabilir. Dilin çevrilmesi sırasında gerek duyulan bazı bilgileri tanımlamada regüler ifade yada gramer kullanılamaz.
- Regüler ifade, DFA ve gramer gibi araçlarla tanımlanamayan bu bilgilere "dilin anlamsal özellikleri" (semantic features) denir.
- Bunlar anlamsal kurallarla belirlenirler. Anlamsal kurallar genellikle sözlü ifadelerle belirlenir.

Anlamsal (semantic) analiz

- Ayrıştırma ağacındaki her düğüm bir gramer simgesine karşı düşer.
- Gramer simgeleri (ağaç düğümleri) kendileriyle ilişkilendirilen niteliklere (tip, adres, bir sayı, bir işaretçi, vs)sahip olabilirler.
- Her gramer kuralının kendisiyle ilişkilendirilmiş olan bir anlamsal kurallar kümesi bulunur. Anlamsal kurallar, o gramer kuralında yer alan simgelere ait niteliklerin ne şekilde hesaplanacağını belirler ve diğer işlemleri (sembol tablosuna bilgi girişi, arakod üretimi, vs)yönlendirirler.

Anlamsal (semantic) analiz

- Anlam çözümleme sonucunda üretilen kod için kullanılan ara diller, genel olarak, üst düzeyli bir birleştirici diline benzerler.
- Bu soyut dil, kaynak dilin veri türleri ve işlemleriyle uyumlu olacak şekilde tasarlanmış, hayali bir makine için bir makine dili olup, derleyicinin kaynak ve amaç dilleri arasında bir ara adım oluşturur.

Kod optimizasyonu

- Üretilen kodun olası çalışma zamanını ya da bu kodu saklamak için gerekli bellek alanını azaltmak için yapılacak iyileştirmeler sürecidir.
- Kodu daha etkin yapmak için, kod parçasında ortak alt ifadelerin kaldırılması, ölü kodun kaldırılması ve çevrim sabitinin çevrim dışına aktarılması gibi bazı iyileştirmeler yapılabilir.

Kod optimizasyonu

- Kaynak programdan soyut programa (ara kod) kadar olan aşamada yapılan iyileştirmeler Makine bağımsız eniyilemelerdir.
- Değerleri bilinen ifadeler derleme zamanında hesaplanırsa aynı hesaplamalar yinelenmez.
- İşlemci yüklemelerinde yapılacak eniyilemeler de (işlemcilerin eşdeğer ama daha hızlı olan işlemcilerle değiştirilmesi) makine bağımsız eniyilemedir.

Kod optimizasyonu

- Çalışma süresini kısaltmak için makina dili özelliklerini kullanarak yapılan eniyilemeler Makine bağımlı eniyilemedir.
- (*increment*) komutunu bir ekleme işlemi yerine kullanmak, makine bağımlı eniyilemeye örnek verilebilir.
- Eniyileme isteğe bağlı bir aşamadır ve eniyileme sonucu, çalıştırılan deyimlerin sırasının kaynak koddaki deyim sıralamasından farklı olabilmesi durumunda hata ayıklama (*debug*) işlemini zora sokabilir.

Kod üretimi

- Bu aşamada donanımdan bağımsız ara koddan hedef donanımın simgesel/makina koduna dönüşümü (Sanal makinadan fiziksel makinaya geçiş) gerçekleştirilir.
- Arakod işlemlerini gerçekleyecek en iyi makina komutlarının seçilmesi gerekir.
- Simgesel dilde kod üretilmiş ise, birleştirici program devreye girer ve makina kodunda hedef program üretir.
- Ara kod üretiminde en sık kullanılan form üçlü adres kodudur.

  a= b operatör c

Kod üretimi

- Ara kod üretiminin direk makine kodunu üretmeye göre bazı avantajları vardır.
  - A adet kaynak dil için b adet makine varsa, a*b adet kod üretecine gerek vardır.
  - Bir makine için yazılan optimizasyon kısmı başka bir makine için kullanılmayacak ve tekrar yazılması gerekecektir. Bu da derleyici tasarımında en zor kısımdır.

Derleyici ve yorumlayıcı karşılaştırılması

- Yorumlayıcıda dilin dönüşümü satır satır gerçekleştiğinden, bir satırın çalıştırılması için gereken çevrim süreci, o satırla ilgili deyimlerin her çalıştırılmasında aynen tekrarlanacaktır.
- LISP dili yorumlayıcı kullanan bir yüksek düzeyli dildir.

Derleyici ve yorumlayıcı karşılaştırılması

- Derleyicide ise program için gerekli makine kodu, deyim kaç kez çalıştırılırsa çalıştırılsın, sadece bir defa bütün olarak oluşturulur.
- Yani bir programın makine koduna çevrimi tamamlandıktan sonra oluşan makine kodu, program kaç kez çalıştırılırsa çalıştırılsın, program değiştirilmediği sürece aynen kullanılabilir.
- Fortran, PASCAL ve C derleyici kullanan yüksek düzeyli dillere örnek olarak verilebilir.

Derleyici ve yorumlayıcı karşılaştırılması

- Hedef kodun oluşturulmasında zaman yönüyle bakıldığında derleme yöntemi, yorumlayıcıya göre zamandan kazanç sağlar.
- Yorumlayıcıda yüksek düzeyli dil deyimleri, orijinal şekillerinde kalırlar ve onların çalıştırılması için gerekli komutlar da yorumlayıcının altprogramları olarak bulunur. Böylece yorumlamayıcı, derleyiciye göre daha az bellek kullanırlar.
- Derleyiciler ve yorumlayıcılar, çalışma sırasındaki hataların kullanıcıya bildirilmesi açısından farklılık gösterirler. Yorumlayıcılar her komutu birer birer ele alırlar. Dolayısıyla tesbit edilen hatalar kullanıcılara doğrudan bildirilir. Derleme yönteminde, tüm program deyimlerinin makine koduna çevrilir ve hatalar toplu olarak bildirilir. Bu durumda, hata kaynağı deyimlerin belirlenmesi, yorumlayıcıya göre zor olmaktadır.