OPERATOR PRECEDENCE PARSING (Operatör Öncelikli Ayrıştırma)

1.Neden Önceliğe İhtiyaç Duyulur?

Birçok matematiksel işlemi barındıran string'lerde hangi işlemin derleyici tarafından önce yapılacağı bilinmezse bir belirsizlik (ambiguity) oluşur. Bu durum karmaşık matematiksel işlemlerin sonuçlarının yanlış çıkmasına sebep olabilir.

```
expr:   expr '-' expr
      | expr '*' expr
      | expr '<' expr
      | '(' expr ')'
      ;
```

Böyle bir gramere sahip olduğumuzu düşünelim, bu gramer, `1 – 2 * 3' girişi için belirsizdir. Çünkü, derleyici bu girişi iki farklı şekilde ayrıştırabilir.

exp = 1-2*3

(1-2*3)

((1-2)*3) = (-1*3) = (-3)

(1-(2*3)

(1-6)

(-5)

Derleyicinin ayrıştırma yaparken böyle durumlarla karşılaşmaması için operatörler arasında öncelik ilişkileri kurulur.

2.Operator Grammar (Operatör Grameri)

Bir gramerin Operator Precedence Parsing yöntemiyle ayrıştırılabilmesi için öncelikle operatör grameri olması gerekmektedir.

Operatör gramerinin temel özellikleri:

- Herhangi bir türetimin sağ tarafında ϵ bulunamaz. Sağ taraftan epsilon olmayacak
- Herhangi bir türetimin sağ tarafında yan yana iki ya da daha fazla nonterminal ifade bulunamaz.

Örnek 1.

E➔EAE | (E) | -E | id

1-2*3

E A E

A➔ + | - | * | / | şeklinde tanımlanan gramer operatör gramer midir?

Hayır. E'nin sağ tarafında birden fazla yan yana nonterminal ifade bulunduğundan bu gramer operatör gramer değildir.

Örnek 2.

E➔ E+E | E-E | E*E | (E) | -E | id şeklinde tanımlanan gramer operatör gramer midir?

Evet. Verilen kurallara uyduğu için bu gramer operatör gramerdir.

yan yana nonterminal iyle

CNBC-e

3.Operatör Oncelikli Ayrıştırma Tekniği ve Özellikleri

Operator Precedence Parsing, Shift-Reduce Parsing işleminin matematiksel ifadeler için uyarlanmış halidir. Shift-Reduce Parsing ise Bottom-Up Parsing'in özel bir halidir.

Bu tekniğin avantajlı ve dezavantajlı yönleri bulunmaktadır. Avantajı, basitliğinden dolayı birçok derleyici tarafından kullanılıyor olmasıdır. Basit işlemlerle çok uzun aritmetik ifadeler ayrıştırılıp doğru bir şekilde sonlandırılabilir. Dezavantajlı yönü ise, eğer tanımlanan dilde aynı operatörün iki farklı önceliği varsa (Örneğin çıkarma ve negatif operatörü) bu durumda parsing işieminin yanlış yapılması olasıdır.

3.1.Operatör Önceliklerini Belirleme

Operatörler arasındaki öncelikleri belirlemek üzere birbirinden farklı üç adet öncelik ilişkisi tanımlanmıştır. Bu ilişki tanımları, iki operatör arasındaki işlem sırasını belirlemek için kullanılırlar ve nonterminaller kümesinde tanımlıdırlar. Derleyici, bu ilişkileri bir tabloda tutarak parsing işlemini bu tablonun yardımıyla yapar.

İlişki	Anlamı
a <. b	a'nın önceliği b'den sonra gelir.(b, a'dan daha önceliklidir.)
a =. b	a ile b eşit önceliğe sahiptir.
a .> b	b 'nin önceliği a'dan sonra gelir.(a, b'den daha önceliklidir.)

Tablo–1

Bu ilişkiler, "küçüktür", "büyüktür", "eşittir" gibi aritmetik ilişki ifadelerine benzemektedir fakat anlamları ve tanımları oldukça farklıdır. Öyle ki:

- Aynı dil için a <. b ve a .> b olabilir.
- a <. b , a .> b , a =. b den hiç biri olmayabilir.

Örnek 3.

* (çarpma) ve – (çıkarma) operatörleri arasındaki öncelik ilişkilerini Tablo-1'e göre tanımlayınız.

Çarpma operatörünün önceliği çıkarma operatöründen yüksek olduğu için bu ilişki, * .> - ve - <. * şeklinde tanımlanmalıdır. Öncelik ilişkisi bu şekilde tanımlandığında bir "operator precedence parser" yaratılabilir. Fakat bu yöntem " - " operatörünün yarattığı karmaşayı çözmez. Bu nedenle iki nonterminal arasında hangi öncelik ilişkisi olması gerektiğine karar vermenin diğer bir yolu gramerdeki belirsizliği (ambiguity) çözmektir.

3.2 Operatör Önceliklerini Stringlere Uygulama

E➔E+E
E➔E*E
E➔(E)
E➔id

	id	+	*	$
id		.>	.>	.>
+	<.	.>	<.	.>
*	<.	.>	.>	.>
$	<.	<.	<.	ACCEPT

Tablo–2

$ <. id .> + <. id .> * <. id .> $

Tanımlanan gramer ve Tablo-2'ye göre id + id * id stringini göz önüne alalım. Bu stringteki operatörlerin öncelik ilişkilerini belirlemek için stringin başını ve sonunu belirleyen bir operatör daha kullanmamız gerekmektedir. Bu operatör "$" işareti ile tanımlanır ve diğer operatörlere göre öncelik durumu Tablo-2'de görülmektedir. Buna göre string $ id+id*id $ haline gelmiştir.

$ id+id*id $ stringini soldan başlayarak Tablo–2 yardımıyla öncelik ilişkilerini gösterecek şekilde yazdığımızda:

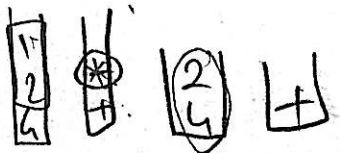
$ <. id .> + <. id .> * <. id .> $ haline gelir.

Bu ifadedeki handle'ı bulmak istersek

- Stringi soldan itibaren ilk .> ifadesini görünceye kadar tararız.
- .> ifadesine rastladıktan sonra geriye doğru ilk <. ifadesine kadar tararız.
- Bu iki ifade arasında kalan ifadeler bu dilin handle'ıdır.

$ <. id .> + <. id .> * <. id .> $ stringinin handle'ı yukarıdaki işlemler uygulandığında id olarak bulunur.

Operator Precedence Parsing işlemi Shift-Reduce Parsing tekniğini temel olarak aldığı için buradaki E➔id production'ı kullanılarak bir reduce işlemi uygulanabilir. Bu işlem uygulandığında string $ E+E*E $ haline gelir.E'li ifadeleri stringten kaldırıp öncelik ilişkilerini yerleştirdiğimizde $ <. + <. * .> $ ifadesini elde ederiz. Elde ettiğimiz ifade nonterminallerden bağımsızdır.



3.3.Operator Precedence Parsing İşlemini Stack Kullanarak Gerçekleştirme

Operatör Precedence Parsing tekniğini kullanan derleyiciler bu işlemi yapabilmek için stack'tan faydalanırlar. Shift-Reduce Parsing tekniğinde kullanılan "Shift", "Reduce", "Accept" ve "Error" işlemleri burada da geçerlidir.

3.3.1.Çift Stack Kullanarak Operator Precedence Parsing İşlemini Gerçekleştirme

Bu yöntemde operatörler ve işlenecek stringte bulunan sayılar için ayrı ayrı iki adet stack tanımlanır. Başlangıçta stack'lar boştur. Sadece stringin başını belirtmesi amacıyla $ operatörü bulunmaktadır. Parse işlemi esnasında bir sayıya rastlandığında bu token, sayılar stack'ına atılır (shift işlemi).Bir operatöre rastlandığında ise stack boş ise bu operatör de stack'a atılır. Fakat stack'a daha önceden atılmış bir operatör varsa bu iki operatör arasındaki öncelik sırası öncelik tablosundan bakılarak belirlenir ve gereken işlem yapılır. Eğer sonradan gelen operatörün önceliği daha düşükse ilk operatör stack'tan pop edilip (reduce işlemi) bu sırada sayılar stack'ında bulunan sayılarla bu işlem gerçekleştirilir. Sonradan gelen operatörün önceliği daha yüksekse shift işlemine devam edilir.

Bu işlem operatör stack'ı boşalıncaya kadar devam eder. Bütün işlemler doğru yapılmış ise sayılar stack'ında işlemin sonucu bulunacaktır.

Adım	Operatör	Sayılar	Giriş	İşlem
1	$	$	4 * 2 + 1 $	shift
2	$	$ 4	* 2 + 1 $	shift
3	$ *	$ 4	2 + 1 $	shift
4	$ *	$ 4 2	+ 1 $	reduce
5	$	$ 8	+ 1 $	shift
6	$ +	$ 8	1 $	shift
7	$ +	$ 8 1	$	reduce
8	$	$ 9	$	accept

Tablo–3

Tablo-3'te yukarıda anlatılan işlemlerin adım adım yapılışı gösterilmiştir. Bu işlemde * operatörünün + operatöründen daha yüksek önceliğe sahip olduğu kabul edilmiştir.

3.3.2.Tek Stack Kullanarak Operator Precedence Parsing İşlemini Gerçekleştirme

Derleyicilerde çoğunlukla kullanılan Operator Precedence Parsing işlemi bu metotla yapılır. Çünkü çift stack kullanmanın maliyeti büyüktür.

E→E+E

E→E*E

E→(E)

E→id ile tanımlanan gramer ve Tablo-2'deki öncelik tablosuna göre id+id*id stringini tek stack ile parse etme işlemleri aşağıda gösterilmiştir.

Parsing işlemi esnasında yapılan production'larla ortaya çıkan E ifadesi "dummy class" olarak kabul edilmiş stack'ta yapılan işlemlerde etkisiz kılınmıştır.

Stack	Giriş Stringi	Sebep	İşlem
$	id + id * id $	---	---
$ id	+ id * id $	$ <. id	shift
$ E	+ id * id $	id .> +	reduce
$ E +	id * id $	$ <. +	shift
$ E + id	* id $	+ <. id	shift
$ E + E	* id $	id .> *	reduce
$ E + E *	id $	+ <. *	shift
$ E + E * id	$	* <. id	shift
$ E + E * E	$	id .> $	reduce
$ E + E	$	* .> $	reduce
$ E	$ ACCEPT!!	+ .> $	reduce

Tablo–4

4..Operator Precedence Parsing Algoritması

Operator Precedence Parsing yöntemini kodlamak istersek aşağıda verilen algoritmayı kullanabiliriz.

Giriş: Giriş stringi (w) ve öncelik ilişkisi tablosu

Çıkış: Eğer w gramer tarafından kabul edilirse parse tree, aksi durumda hata mesajı

Başlangıçta stack $ içerir ve giriş buffer'ında w$ var.

Ip' yi w$ stringinin ilk sembolünü gösterecek şekilde ayarlıyoruz

repeat forever

  if $ stack'ın üstünde ve ip $ a işaret ediyorsa **then**

    return

  else begin

    a stack'ın üstündeki terminali ve b ip tarafından işaret edilen sembolü göstersin

    if a <. b veya a =. b **then begin** /* shift */

      b'yi stack'a it (push) ve ip'yi bir sonraki sembolü gösterecek şekilde düzenle

  end

  else if a .> b **then** /* reduce */

    repeat

      stack'ta pop işlemi yap

    until stack'in üstündeki nonterminal <. en son stack'tan çekilen nonterminal

  else error()

end

5.Associativity ve Operator Precedence Kuralları

Bir derleyici tasarlarken o derleyicinin kullanacağı operatör önceliği ilişkileri belirli kurallarla tanımlanmalıdır. Bu kurallar öncelikle doğru handle'ı bulmak ve grameri doğru oluşturmak amacıyla belirlenmişlerdir.

1. Eğer Q1, Q2'den daha yüksek önceliğe sahipse Q1 .> Q2 ve Q2 <. Q1 olarak tanımlanır. Örneğin / , -'den daha öncelikli ise / .> - ve - <. / olur. Bu kural, E – E / E – E şeklindeki bir ifadede handle'ın E / E olduğunu ve ilk önce bu ifadenin işlem göreceğini ispatlar.
2. Eğer Q1 ve Q2 eşit önceliğe sahip ise (Q1 ve Q2 aynı operatörler olabilir)

   Q1 ve Q2 left-associative ise

     Q1 .> Q2 ve Q2 .> Q1 olur.(Öncelikli olanlar sol tarafta)

   Q1 ve Q2 right-associative ise

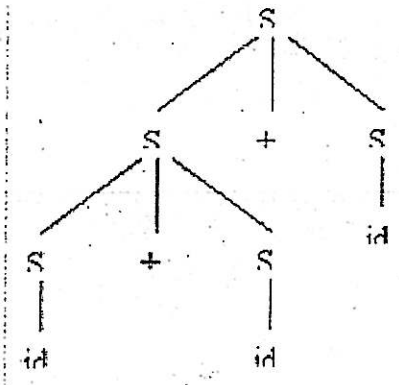
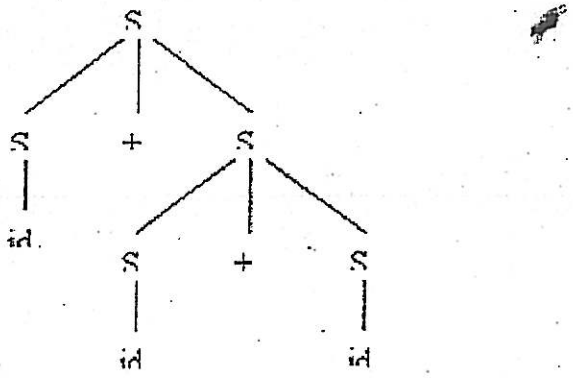
     Q1 <. Q2 ve Q2 <. Q1 olur.(Öncelikli olanlar sağ tarafta)

3.Her Q operatörü için Q <. id ve id .> Q olur. Ayrıca her Q operatörü için Q .> $ ve $ <. Q 'dır.Diğer önemli operatör ilişkileri:

(=.)	$ <. (	$<.id
(<. (	id .> $	) .> $
(<. id	id .>)	) .>)

5.1 Left-Associative ve Right-Associative

Associative, aralarında ilişki kurmak anlamına gelmektedir. Operatörler arasında ilişki kurulurken bu ilişkinin soldan itibaren mi yoksa sağdan itibaren mi düşünüleceği belirtilmelidir. Çünkü, oluşturulacak parse ağacı işlemlerin left-associative ya da right-associative olmasına göre değişmektedir. Örneğin id + id + id ifadesinin left ya da right associative olmasına göre değişen parse ağaçları aşağıda gösterilmiştir.

"+" operatörü Left Associative ise	**"+" operatörü Right Associative ise**
(id + id) + id	id + (id + id)
	

Örnek 4. E ➔ E+E|E–E|E * E|E / E|E ↑ E|(E)|-E|**id** grameri için su özellikler tanımlanıyor:

1. ↑ en yüksek önceliğe sahip ve right-associative
2. * ve / sonraki en yüksek önceliğe sahip ve left-associative
3. + ve – en düşük önceliğe sahip ve left associative

id * (id ↑ id) – id / id stringinin parsing adımlarını gösteriniz.(Tablo–5 yardımıyla)

	+	−	*	/	↑	id	(	)	$
+	>	>	<	<	<	<	<	>	>
−	>	>	<	<	<	<	<	>	>
*	>	>	>	>	<	<	<	>	>
/	>	>	>	>	<	<	<	>	>
↑	>	>	>	>	<	<	<	>	>
id	>	>	>	>	>			>	>
(	<	<	<	<	<	<	<	=	
)	>	>	>	>	>			>	>
$	<	<	<	<	<	<	<		

Tablo–5

E → E+E | E−E | E*E | E/E | E↑E | (E) | −E | id

Tablo–5 'te görülen öncelik tablosuna göre verilen string Tablo–6 'da görüldüğü şekilde parse edilir.

Stack	Giris
$	id * (id ↑ id) − id / id $
$ id	* (id ↑ id) − id / id $
$	* (id ↑ id) − id / id $
$ *	(id ↑ id) − id / id $
$ * (	id ↑ id) − id / id $
$ * (id	↑ id) − id / id $
$ * (	↑ id) − id / id $
$ * (↑	id) − id / id $
$ * (↑ id	) − id / id $
$ * (↑	) − id / id $
$ * (	) − id / id $
$ * ( )	− id / id $
$ *	− id / id $
$	− id / id $
$ -	id / id $
$ - id	/ id $
$ -	/ id $
$ - /	id $
$ - / id	$
$ - /	$
$ -	$
$	$

Tablo–6

6.Tekli Operatörlerin Önceliği

Tekli operatörlerin diğer operatörlerde olduğu gibi öncelik ilişkileri tanımlanabilir. Örneğin Γ operatörü için Q, herhangi bir operatör ve Γ, Q'dan daha yüksek önceliğe sahip ve left-associative ise

E & Γ E & E ➔ (E & (Γ E)) & E yazılabilir.

Eğer bir operatör hem tekli hem de ikili operatör ise ve bu iki operatöre eşit öncelik verilmişse bazı işlemler için doğru parse ağacı elde edilemez.(Örneğin id*-id)Bu durumda en iyi yaklaşım, bu iki operatörü ayırmak için lexical analyzer kullanmaktır. Lexical analyzer bu iki operatör için ayrı ayrı token'lar üretirse bu karışıklık çözülür fakat bu imkânsızdır. Bu nedenle böyle durumlarda bir önceki operatöre bakılır. id * - id örneği için eğer bir önceki operatör (= ya da , ise işlem negatifini alma işlemi olarak alınır. Bu kurallar derleyicinin tasarımına göre değişir.

7. Öncelik Fonksiyonları

Operator Precedence Parser kullanan derleyiciler operatör önceliklerini belirleyen tabloları tutmak yerine bu tabloların kodlanmış hali üzerinde çalışan fonksiyonlar kullanırlar. Terminal sembollerinden integer sayılara tanımlı f ve g öncelik fonksiyonları, terminal sembollerini giriş olarak kabul edip sonucunda integer bir sayı üretirler. Çıkışlar karşılaştırılıp hangi terminal sembolünün daha öncelikli olduğuna karar verilir.

f ve g' nin tanımı aşağıdaki gibidir

- f(a) < g(b) eğer a <. b ise
- f(a) = g(b) eğer a =. b ise
- f(a) > g(b) eğer a .> b ise

a ve b terminalleri arasındaki precedence ilişkisi f(a) ve f(b) nin kıyaslanmasıyla elde edilir.

Tablo-5'e göre hazırlanan öncelik fonksiyonları Tablo-7'de gösterilmiştir:

	+	_	*	/	↑	(	)	id	$
f	2	2	4	4	4	0	6	6	0
g	1	1	3	3	5	5	0	5	0

Tablo–7

* <. id olduğundan dolayısıyla f(*) = 4 < g(id) = 5 olur.

7.1 Öncelik Fonksiyonlarının Elde Edilmesi

Bir öncelik tablosundan, öncelik fonksiyonlarını üretmek için gerekli adımlar aşağıda açıklanmıştır:

Giriş: Operatör önceliği tablosu

Çıkış: Girişi temsil eden çıkış fonksiyonları ya da bu fonksiyonların bulunamayacağının ifadesi

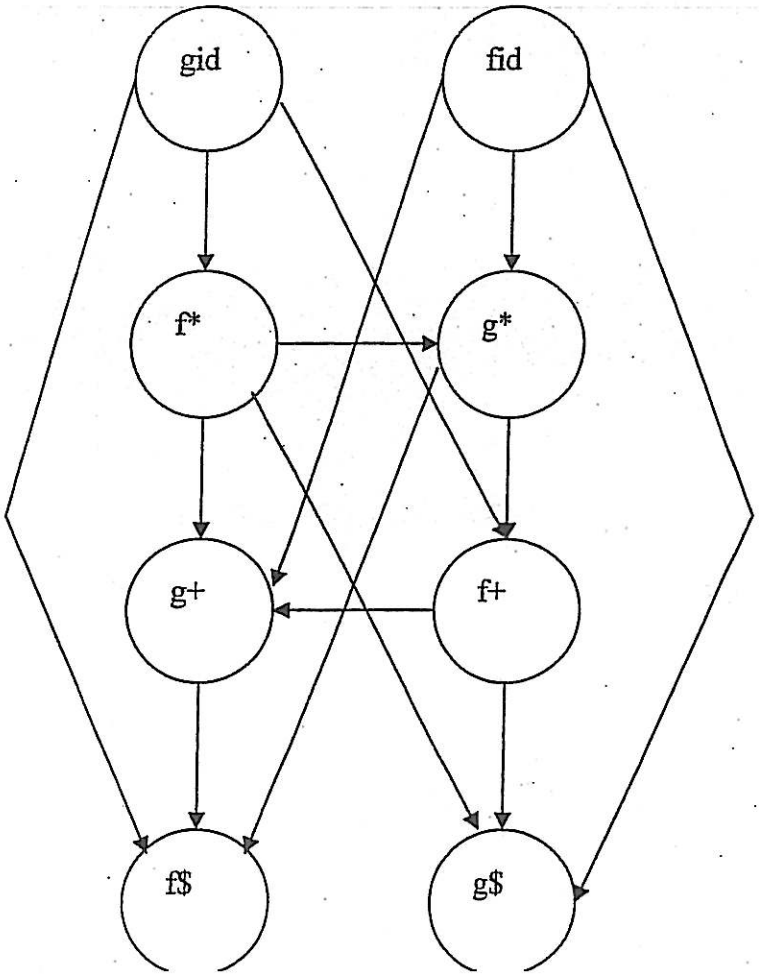
1. Her bir terminal için (örneğin a) fa ve ga olmak üzere iki sembol oluştur.

2. Eğer a =. b ise fa ve fb ayni grupta olmak şartıyla sembolleri maksimum sayıda gruba ayır.

3. a ve b için eğer a < . b ise gb nin bulunduğu gruptan fa nın bulunduğa gruba bir kenar olan; eğer a .>b ise fa nın bulunduğu gruptan gb nin bulunduğu gruba bir kenar olan yönlü bir graf oluştur.

4. Eğer elde edilen graf bir döngü oluşturuyorsa, öncelik fonksiyonları bulunamaz (yoktur). Aksi takdirde f(a), grup fa dan başlayan en uzun yolun uzunluğudur, g(a) ise grup ga dan başlayan en uzun yolun uzunluğudur.

Tablo-2'deki öncelik tablosunun öncelik fonksiyonlarını elde edecek olursak, Tablo-8'deki değerleri buluruz.



	+	*	id	$
f	2	4	4	0
g	1	3	5	0

Tablo–8