**Operator precedence parsing**

An operator precedence parser is a bottom-up parser that interprets an operator grammar.

Ambiguous grammars are not allowed in any parser except operator precedence parser.

This parser is only used for operator grammars.

A grammar is said to be operator precedence grammar if it has two properties:

* No R.H.S. of any production has a∈. A-> ε
* No two non-terminals are adjacent. A->BC

E->E+E/E\*E/id

E->EAE |A

A->+ | \* | / | ↑

E->E+E |E\*E | E/E |E↑E

S->SAS/a

A->bSb/b

S-> SbSbS | SbS | a

## There are the three operator precedence relations:

Operator precedence can only established between the terminals of the grammar. It ignores the non-terminal.

**a ⋗ b** means that terminal "a" has the higher precedence than terminal "b".

**a ⋖ b** means that terminal "a" has the lower precedence than terminal "b".

**a ≐ b** means that the terminal "a" and "b" both have same precedence.

There are two methods for determining what precedence relations should hold between a pair of terminals:

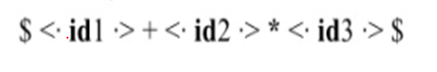
1. Use the conventional associativity and precedence of operator.
2. The second method of selecting operator-precedence relations is first to construct an unambiguous grammar for the language, a grammar that reflects the correct associativity and precedence in its parse trees.

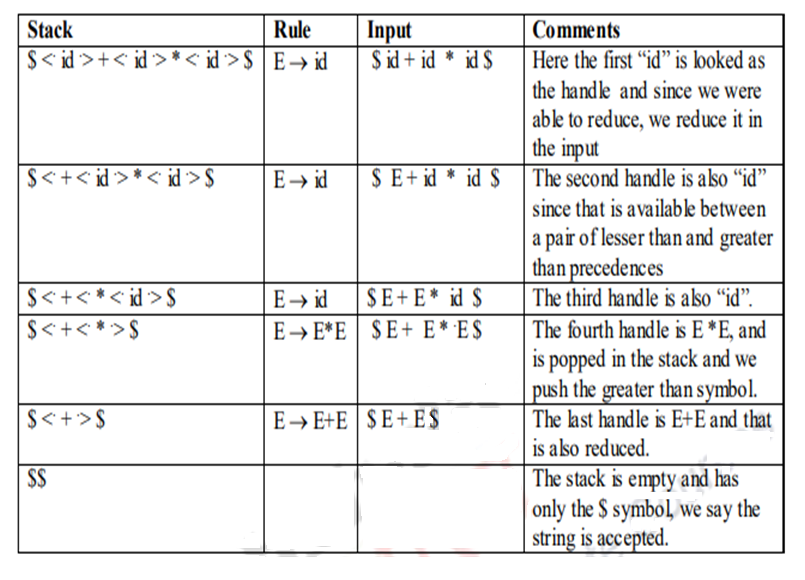
Grammar:

E->E+E | E\*E | id id+id\*id+id



String: $id+id\*id$



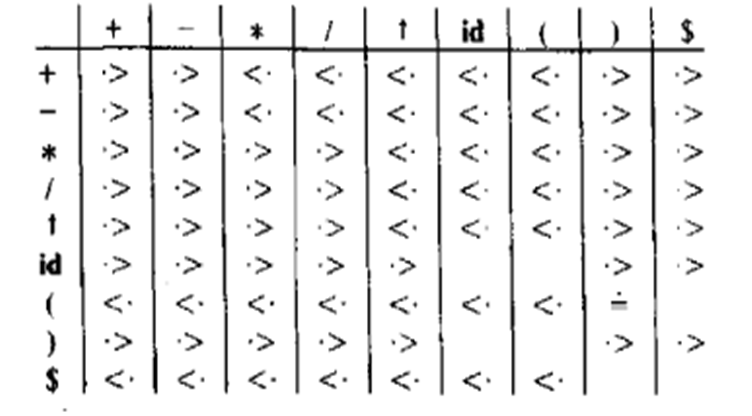


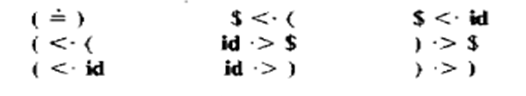
a+(b+c)/d-e+a

c=a(b+c)

a=((b+c)(c\*d)\*d

a+(a+b$)





**Operator-Precedence Parser**

Computation of the function LEADING() and TRAILING()

Leading(A)

1. If A->γaδ , where γ is a single Non Terminal or ε

Then Leading(A)={a}

1. If A->Bα , then

Leading(A)=Leading(B)

S->a | ↑ | (T)

T-> T,S | S

Leading(S)={a, ↑, ( }

Leading(T)={ , , Leading(T), Leading(S)}={, , a, ↑, ( }

S-> L=R

S->R

L->\*R

L->id

R->L

Leading(S)={=,Lead(L),Lead(R )}={=,\*,id}

Leading(L)={\*,id}

Leading(R)={Lead(L)}={\*,id}

Trailing(A)

1. If A->αaβ , where β is a single variable(NT) or ε

Trailing(A)={a}

1. If A->αaB ,

Trailing(A)= Trailing(B)

S->a | ↑ | (T)

T-> T,S | S

Trailing (S)={a, ↑,)}

Trailing (T)={, , Trailing(S)} ={,, a, ↑,)}

S-> L=R

S->R

L->\*R

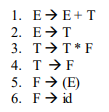
L->id

R->L

Trailing (S)={=,Trailing( R)}={=,\*,id}

Trailing (L)={\*,id, Trailing( R)}={\*,id}

Trailing (R)={Trailing(L)}={\*,id}



1. Check whether the given grammar is an operator grammar
2. Compute the Leading and Trailing for all non terminals

Leading( E)={+,\*,(,id} Trailing( E)={+,\*,id,)}

Leading( T)={\*,(,id} Trailing( T)={\*,id,)}

Leading( F)={(,id} Trailing( F)={id,)}

1. Construction of Operator Precedence Relation Table
2. $ <**.** Leading(S) $ <**.{**+,\*,(,id}

Trailing(S) **.**> $ {+,\*,id,)}>$

1. T NT T

T T (=)

1. T NT

T<**.** Leading(NT)

1. NT T E+ Trail( E)>+ {+,\*,id,)}>+

Trailing(NT) **.**> T T\* {\*,id,)}>\*

E) {+,\*,id,)}>)

1. T T

T T c=a(b+c)(d\*c)($)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | + | \* | Id | ( | ) | $ |
| + | > | < | < | < | > | > |
| \* | > | > | < | < | > | > |
| Id | > | > |  |  | > | > |
| ( | < | < | < | < | = |  |
| ) | > | > |  |  | > | > |
| $ | < | < | < | < |  | Accept |

|  |  |  |  |
| --- | --- | --- | --- |
| Stack Content | Input | Relation | Action |
| $ | (id+id)\*id$ |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

1. Consider the following grammar, and construct the operator precedence parsing table and check whether the input string (i) \*id=id (ii)id\*id=id are successfully parsed or not?

**S→L=R**

**S→R**

**L→\*R**

**L→id**

**R→L**

Leading(S)={=,Lead(L),Lead(R )}={=,\*,id}

Leading(L)={\*,id}

Leading(R)={Lead(L)}={\*,id}

Trailing (S)={=,Trailing( R)}={=,\*,id}

Trailing (L)={\*,id, Trailing( R)}={\*,id}

Trailing (R)={Trailing(L)}={\*,id}

### Precedence Table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | = | \* | id | $ |
| = |  |  |  |  |
| \* |  |  |  |  |
| id |  |  |  |  |
| $ |  |  |  |  |

\* All undefined entries are error (e).

### Parsing the given input string:

### \*id = id

|  |  |  |  |
| --- | --- | --- | --- |
| **STACK** | **INPUT STRING** | **ACTION** | |
| $ | \*id=id$ |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

* + 1. **id\*id=id**

|  |  |  |
| --- | --- | --- |
| **STACK** | **INPUT STRING** | **ACTION** |
| $ | id\*id=id$ |  |
|  |  |  |

**Precedence Functions**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **+** | **\*** | **id** | **$** |
| **+** | **>** | **<** | **<** | **>** |
| **\*** | **>** | **>** | **<** | **>** |
| **id** | **>** | **>** | **e** | **>** |
| **$** | **<** | **<** | **<** | **Accept** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **id** | **+** | **\*** | **$** |
| f |  |  |  |  |
| g |  |  |  |  |

2. Consider the following grammar, and construct the operator precedence parsing table and check whether the input string **(a,(a,a))** is successfully parsed or not?

S-> (L) | a L-> L,S | S

( i) Check whether the grammar is an operator grammar

(ii) Computation of Leading and Trailing

Leading(S)={(,a}

Leading(L)={, , Leading(S)}={, , (,a}

Trailing(S)={), a}

Trailing(L)={, , Trailing(S)}= {, , ), a}

1. Construction of operator relation table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | a | , | ( | ) | $ |
| a |  | > |  | > | > |
| , | < | > | < | > | > |
| ( | < | < | < | = |  |
| ) |  | > |  | > | > |
| $ | < |  | < |  |  |

1. $< Leading(S) L) => Trailing(L)>) L, Trail(L)>,

$ < {(,a}

Trailing(S)>$

{), a}>$

T1 NT T2

T1=T2

(L)

(=)

T NT

T < Leading(NT)

( L

(<Leading(L)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | a | , | ( | ) | $ |
| a | e | > | e | > | > |
| , | < | > | < | > | > |
| ( | < | < | < | = | e |
| ) | e | > |  | > | > |
| $ | < |  | < |  | Accept |

1. Parse the string: (a,(a,a))

|  |  |  |
| --- | --- | --- |
| **Stack** | **Input** | **Action** |
| **$** | **(a,(a,a))$** | $<( Push ( |
| $( | **a,(a,a))$** | S<a Push a |
| $(a | **,(a,a))$** | a>, Pop a |
| $( | **,(a,a))$** | (<, Push , |
| $(, | (a,a))$ | ,<( Push ( |
| $(,( | a,a))$ | (<a Push A |
| S(,(a | ,a))$ | Pop a |
| $(,( | ,a))$ | Push , |
| $(,(, | a))$ | Push a |
| $(,(,a | ))$ | Pop a |
| $(,(, | ))$ | Pop , |
| $(,( | ))$ | Push ) |
| $(,() | )$ | Pop ) |
| $(,( | )$ | Pop ( |
| $(, | )$ | Pop , |
| $( | )$ | Push ) |
| $() | $ | Pop) |
| $( | $ | Pop ( |
| $ | $ | Accept |