Recursive descent parser (For every variable, there will be a f^n) → after backtracking ~~again~~

$E \rightarrow iE'$

$E' \rightarrow +iE' / \epsilon$

l → lookahead variable

```
E()        l = getchar();
{
1  if (l == i)
2  {  match('i');
3     E'();
   }
}
```

```
E'()
{
  if (l == '+')
  {
    match('+');
    match('i');
    E'();
  }
  else
    return;   // for ε production
}
```

```
match (char t)
{
  if (l == t)
     l = getchar();
  else
     printf("error");
}
```

```
main()
{
1  E();
2  if (l == '$')
3  pf("parsing successful");
}
```

Eg: Generate: i + i $

main() E()

1 | 2 | 3

~~Matched~~

Gen: i + i $

Bottom up Parser
Operator Precedence Parsing (No prob with ambiguity)

Operator precedence parser - Operator grammar

Operator grammar - generally defines mathematical grammar.

(1) $E \to E+E \,/\, E*E \,/\, id$

- (a) No 2 variables are adjacent
- (b) No ϵ production.

} Operator grammar

convert

$E \to EAE \,/\, id$
$A \to + \,/\, *$

} Not operator grammar.

adjacent

In general, in any mathematical expr i.e. two identifiers don't come together.

(2) $S \to SAS/a$
$A \to bSb/b$

$\Rightarrow$ Not operator grammar.

Expand $S \to SbSbS \,/\, SbS \,/\, a$
$A \to bSb \,/\, b$

} No 2 var are adjacent.

$\to$ Using operator relatn table the parser works.

$E \rightarrow E+E \,/\, E*E \,/\, id$

$\cdot>$ higher precedence

$<\cdot$ lower "

Operation relation table has to be constructed
or
i.e. precedence "

	id	+	*	$
id	—	$\cdot>$	$\cdot>$	$\cdot>$
+	$<\cdot$	$\cdot>$	$<\cdot$	$\cdot>$
*	$<\cdot$	$\cdot>$	$\cdot>$	$\cdot>$
$	$<\cdot$	$<\cdot$	$<\cdot$	—

identifier will have higher precedence than any operator.

\+ → left associative ∴ we consider the higher precedence for left +.

$ → least precedence compared to any operator

* → left associative

Similar to infix to postfix conversion

Eg. Parse id + id * id \$

Whenever top of the stack is $\le$ lookahead $\Rightarrow$ push

top " " " $\geq$ lookahead $\Rightarrow$ pop

\$ id + id * id \$ $\uparrow$ | \$ |

\$ $\le$ id $\Rightarrow$ push id

id + id * id \$ $\uparrow$ | \$ | id |

id $\geq$ + $\Rightarrow$ \$ pop id $\Rightarrow$ id is reduced to E in the tree

\$ $\le$ + $\Rightarrow$ push + | \$ | + |

id + id * id \$ $\uparrow$

+ $\le$ id $\Rightarrow$ push id | \$ | + | id |

id + id * id \$ $\uparrow$

id $\geq$ * $\Rightarrow$ pop id $\rightarrow$ id is reduced to E

+ $\le$ * $\Rightarrow$ push * | \$ | + | * |

id + id * id \$ $\uparrow$

* $\le$ id $\Rightarrow$ push id | \$ | + | * | id |

id + id * id \$ $\uparrow$

\$ id $\geq$ \$ pop id $\Rightarrow$ id to E | \$ | + | * |

* $\geq$ \$ pop * | \$ | + |

+ $\geq$ \$ pop + | \$ |

\$ \$ $\rightarrow$ Success

Q2) id + id + id

$ < id push id $\boxed{\$ \mid id}$

id + id + id

id > + pop id $\boxed{\$ \mid +}$

$ < + push +

id + id + id

+ < id push id $\boxed{\$ \mid + \mid id}$

id + id + id

id > + pop id

$\boxed{\$ \mid +}$

+ and + left most + > right most +

+ > +

pop +

id + id + id $

$ < + push +

$\boxed{\$ \mid +}$

id + id + id $

+ < id $ push id

$\boxed{\$ \mid + \mid id}$

id + id + id $

id > $ → pop id

+ > $ → pop +

$ $ ⇒ Success

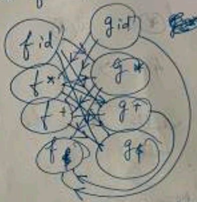
If there are n operators in the grammar the no of entries $\Longrightarrow \boxed{O(n^2)} \Rightarrow$ size of the relation table

∴ Disadvantage = size of the table.

In order to ↓ the size of the table ⟹ we create operator f^n table

ie for rows take f and column g

f \ g	id	+	*	$
id	–	>	>	>
+	<	>	<	>
*	<	>	>	>
$	<	<	<	–



> from f → g arrow

< f ← g

Using the graph find the length of the longest path

After constructing the graph, check if it is cyclic, if yes the operator f^n table can't be generated

Find the length of longest path starting from a particular node.

$$f_{id} \longrightarrow g_* \longrightarrow f_+ \longrightarrow g_+ \longrightarrow f_\$$$

$$g_{id} \longrightarrow f_* \longrightarrow g_* \longrightarrow f_+ \longrightarrow g_+ \longrightarrow f_\$$$

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

Func^n table size $O(2n)$

$f(+)$ $g(+)$

$2 > 1$ $+ > +$

$f(*)$ $g(+)$

$4 > 3$ $* > +$

$f(*)$ $g(*)$

$4 > 1$ $* > *$

Same results can be obtained, with lesser size of table

Disadvantage :-

id and id $\Rightarrow$ Blank in relatn table

$\Rightarrow$ Non-blank entry in fn table.

Blank entry is generally error.

$\therefore$ the error detectn capability of ~~rel~~ fn table is lesser than that of relatn table.

But generally, fn table is used.

In C, no. of operators - approx 100

relatn table $\rightarrow$ 10000 entries

fn " $\rightarrow$ 200 "

Handle Pruning?

Reduct^n

LR

LR(0)

SLR

LALR

CLR