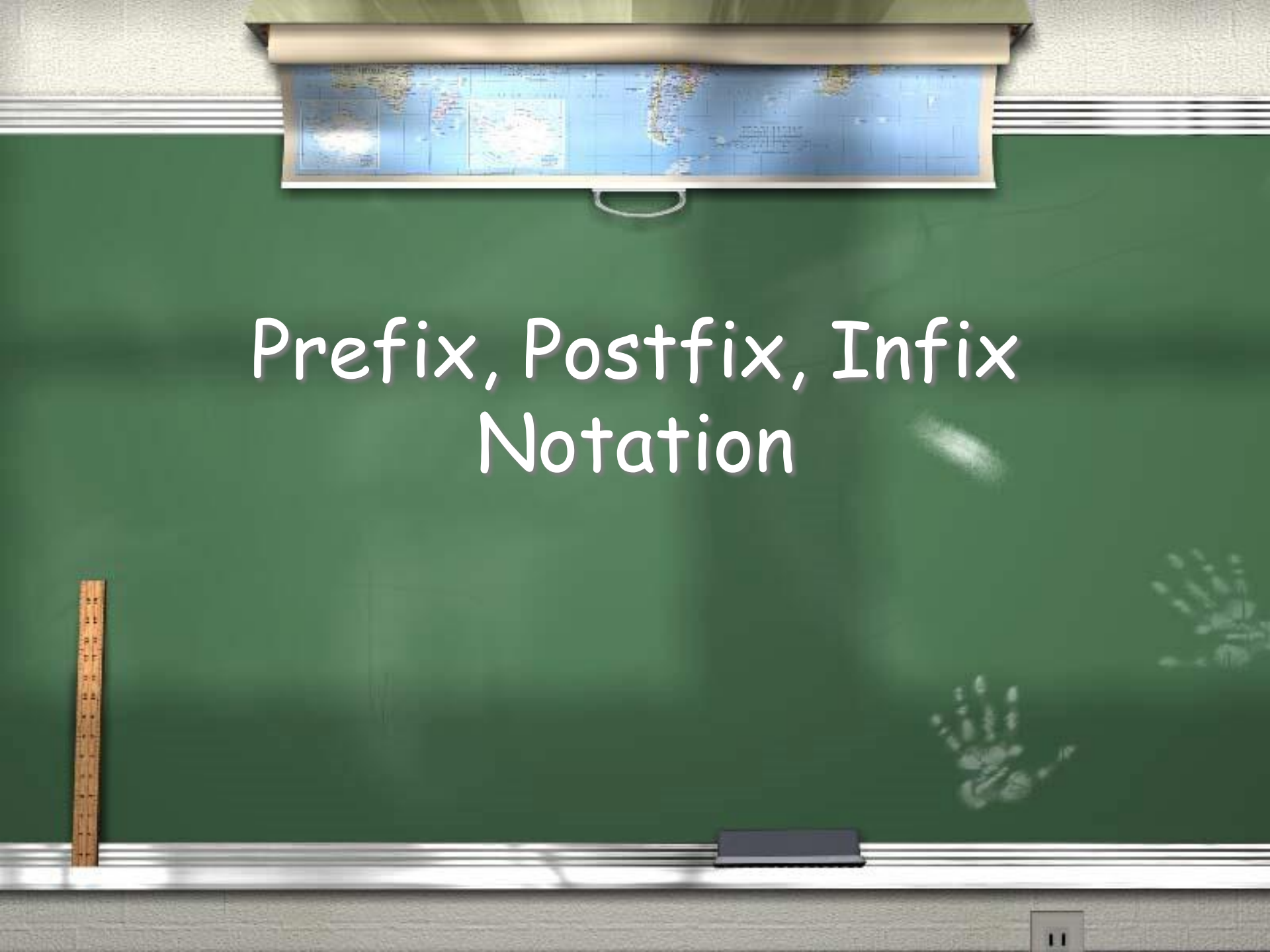

Prefix, Postfix, Infix
Notation

Infix Notation

- To add A, B, we write

  A+B

- To multiply A, B, we write

  A*B

- The operators ('+' and '*') go in between the operands ('A' and 'B')
- This is "*Infix*" notation.

Prefix Notation

- Instead of saying "A plus B", we could say "add A,B " and write

  + A B

- "Multiply A,B" would be written

  * A B

- This is *Prefix* notation.

Postfix Notation

- Another alternative is to put the operators after the operands as in

A B +

and

A B *

- This is *Postfix* notation.

Pre A In B Post

- The terms infix, prefix, and postfix tell us whether the operators go between, before, or after the operands.

Parentheses

- Evaluate 2+3*5.
- + First:

  (2+3)*5 = 5*5 = 25
- * First:

  2+(3*5) = 2+15 = 17
- Infix notation requires Parentheses.

What about Prefix Notation?

- + 2 * 3 5 =
  - = + 2 <u>* 3 5</u>
  - = <u>+ 2 15</u> = 17
- * + 2 3 5 =
  - = * <u>+ 2 3</u> 5
  - = <u>* 5 5</u> = 25
- No parentheses needed!

Postfix Notation

- 2 3 5 * + =
  - = 2 3 5 * +
  - = 2 15 + = 17
- 2 3 + 5 * =
  - = 2 3 + 5 *
  - = 5 5 * = 25
- No parentheses needed here either!

Conclusion:

- Infix is the only notation that requires parentheses in order to change the order in which the operations are done.

Fully Parenthesized Expression (FPE)

- A FPE has exactly one set of Parentheses enclosing each operator and its operands.
- Which is fully parenthesized?

(A + B) * C

★ ((A + B) * C)

((A + B) * (C))

Infix to Prefix Conversion

Move each operator to the left of its operands & remove the parentheses:

((A + B) * (C + D))

Infix to Prefix Conversion

Move each operator to the left of its operands & remove the parentheses:

(+ A B * (C + D))

Infix to Prefix Conversion

Move each operator to the left of its operands & remove the parentheses:

* + A B (C + D)

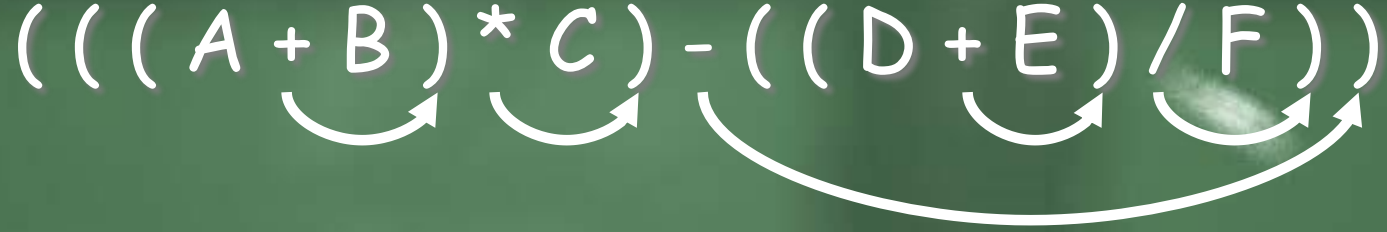
Infix to Prefix Conversion

Move each operator to the left of its operands & remove the parentheses:

* + A B + C D

Order of operands does not change!

Infix to Postfix



A B + C * D E + F / -

- Operand order does not change!
- Operators are in order of evaluation!

Computer Algorithm FPE Infix To Postfix

- Assumptions:

1. Space delimited list of tokens represents a FPE infix expression
2. Operands are single characters.
3. Operators +,-,*,/

FPE Infix To Postfix

- Initialize a Stack for operators, output list
- Split the input into a list of tokens.
- for each token (left to right):
  - if it is operand: append to output
  - if it is '(': push onto Stack
  - if it is ')': pop & append till '('

FPE Infix to Postfix

(((A+B)*(C-E))/(F+G))

- stack: <empty>
- output: []

FPE Infix to Postfix

((A+B)*(C-E))/(F+G))

- stack: (
- output: []

FPE Infix to Postfix

(A + B) * (C - E)) / (F + G))

- stack: ((
- output: []

FPE Infix to Postfix

A + B) * (C - E)) / (F + G))

- stack: (((
- output: []

FPE Infix to Postfix

+ B) * (C - E)) / (F + G))

- stack: (((
- output: [A]

FPE Infix to Postfix

B) * (C - E)) / (F + G))

- stack: (((+
- output: [A]

FPE Infix to Postfix

) * (C - E)) / (F + G))

- stack: (((+
- output: [A B]

FPE Infix to Postfix

* (C - E)) / (F + G))

- stack: ((
- output: [A B +]

FPE Infix to Postfix

(C - E)) / (F + G))

- stack: ((*
- output: [A B +]

FPE Infix to Postfix

C - E)) / (F + G))

- stack: ((* (
- output: [A B +]

FPE Infix to Postfix

- E)) / (F + G))

- stack: ((* (
- output: [A B + C]

FPE Infix to Postfix

E)) / (F + G))

- stack: ((* (-
- output: [A B + C]

FPE Infix to Postfix

)) / (F + G))

- stack: ((* (-
- output: [A B + C E]

FPE Infix to Postfix

) / (F + G))

- stack: ((*
- output: [A B + C E -]

FPE Infix to Postfix

/ (F + G))

- stack: (
- output: [A B + C E - *]

FPE Infix to Postfix

(F + G))

- stack: (/
- output: [A B + C E - *]

FPE Infix to Postfix

F + G))

- stack: (/ (
- output: [A B + C E - *]

FPE Infix to Postfix

+ G))

- stack: (/ (
- output: [A B + C E - * F]

FPE Infix to Postfix

G))

- stack: (/ (+
- output: [A B + C E - * F]

FPE Infix to Postfix

))

- stack: (/ (+
- output: [A B + C E - * F G]

FPE Infix to Postfix

)

- stack: (/
- output: [A B + C E - * F G +]

FPE Infix to Postfix

- stack: <empty>
- output: [A B + C E - * F G + /]

Problem with FPE

- Too many parentheses.
- Need to establish precedence rules.
- We can alter the previous program to use the precedence rules.

Infix to Postfix

```
while there are more symbols to be read
    read the next symbol
    case:
        operand   -->  output it.
          '('     -->  push it on the stack.
          ')'     -->  pop operators from the stack to the output
                       until a '(' is popped; do not output either
                       of the parentheses.
        operator  -->  pop higher- or equal-precedence operators
                       from the stack to the output; stop before
                       popping a lower-precedence operator or
                       a '('.  Push the operator on the stack.
    end case
end while
pop the remaining operators from the stack to the output
```