<table>
<thead>
<tr>
<th>Description</th>
<th>Syntax</th>
<th>Remarks and examples</th>
<th>Conformability</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>++i and i++ increment i; they perform the operation ( i = i + 1 ). ++i performs the operation before the evaluation of the expression in which it appears, whereas i++ performs the operation afterward.</td>
<td>++i increment before</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--i and i-- decrement i; they perform the operation ( i = i - 1 ). --i performs the operation before the evaluation of the expression in which it appears, whereas i-- performs the operation afterward.</td>
<td>--i decrement before</td>
<td></td>
<td></td>
</tr>
<tr>
<td>++i and i++ increment i; they perform the operation ( i = i + 1 ). ++i performs the operation before the evaluation of the expression in which it appears, whereas i++ performs the operation afterward.</td>
<td>i++ increment after</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--i and i-- decrement i; they perform the operation ( i = i - 1 ). --i performs the operation before the evaluation of the expression in which it appears, whereas i-- performs the operation afterward.</td>
<td>i-- decrement after</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Syntax**

++i increment before
--i decrement before
i++ increment after
i-- decrement after

where i must be a real scalar.

**Remarks and examples**

These operators are used in code, such as

\[
\begin{align*}
x[i++] &= 2 \\
x[-i] &= 3 \\
\text{for (i=0; i<100; i++)} &\{ \\
\quad \ldots \\
\text{if (++n > 10)} &\{ \\
\quad \ldots \\
\}
\end{align*}
\]

Where these expressions appear, results are as if the current value of i were substituted, and in addition, i is incremented, either before or after the expression is evaluated. For instance,

\[
x[i++] = 2
\]

is equivalent to

\[
x[i] = 2 \; ; \; i = i + 1
\]

and

\[
x[++i] = 3
\]
is equivalent to
\[ i = i + 1 \; ; \; x[i] = 3 \]

Coding
```c
for (i=0; i<100; i++) {
    ...
}
```
or
```c
for (i=0; i<100; ++i) {
    ...
}
```
is equivalent to
```c
for (i=0; i<100; i=i+1) {
    ...
}
```
because it does not matter whether the incrementation is performed before or after the otherwise null expression.
```c
if (++n > 10) {
    ...
}
```
is equivalent to
```c
n = n + 1
if (n > 10) {
    ...
}
```
whereas
```c
if (n++ > 10) {
    ...
}
```
is equivalent to
```c
if (n > 10) {
    n = n + 1
    ...
}
else n = n + 1
```
The `++` and `--` operators may be used only with real scalars and are usually associated with indexing or counting. They result in fast and readable code.
Conformability

\[ \text{++}, \text{--}, \text{i++}, \text{and } \text{i--}: \]
\[
i: \quad 1 \times 1
\]
\[
\text{result: } \quad 1 \times 1
\]

Diagnostics

++ and -- are allowed with real scalars only. That is, \text{++} \text{i} or \text{i++} is valid, assuming \text{i} is a real scalar, but \text{x[i,j]}++ is not valid.

++ and -- abort with error if applied to a variable that is not a real scalar.

++i, i++, --i, and i-- should be the only reference to \text{i} in the expression. Do not code, for instance,

\[
x[i++] = y[i]
\]
\[
x[++i] = y[i]
\]
\[
x[i] = y[i++]
\]
\[
x[i] = y[++i]
\]

The value of \text{i} in the above expressions is formally undefined; whatever is its value, you cannot depend on that value being obtained by earlier or later versions of the compiler. Instead code

\[
i++ ; x[i] = y[i]
\]
or code

\[
x[i] = y[i] ; i++
\]

according to the desired outcome.

It is, however, perfectly reasonable to code

\[
x[i++] = y[j++]
\]

That is, multiple ++ and -- operators may occur in the same expression; it is multiple references to the target of the ++ and -- that must be avoided.

Also see

[M-2] \textbf{exp} — Expressions

[M-2] \textbf{Intro} — Language definition