Range Definitions

integer range [1..5] one_five;

Indicates input parameters to the current semantic procedure.
Range Definitions

integer range [1..5] one_five;

newType
go

range 2
lo=1
hi = 5
bo =4
base =

ConstExp 5
ConstExp 1
type

int 2

DoMakeRange

C1
...
...int 1
int 5
integer range [1..5] one_five;

newType

range 2
lo=1
hi = 5
bo = 4
base =

return value

int 2

DoMakeRange
Array Definitions

integer array [one_five] [one_three] *a, b;

Assumes we have previously defined
integer range [1..5] one_five;
integer range [1..3] one_three;

DoMakeArray

vector

range 1 3...
range 1 5...
int 2
Array Definitions

integer array [one_five] [one_three] *a, b ;

in DoMakeArray
Array Definitions

integer array [one_five] [one_three] *a, b;

Work backwards thru the vector (last to first)

currentType

finalType

array 6
subs=
comp=

range 1
3...

in DoMakeArray
Array Definitions

integer array [one_five] [one_three] *a, b;

At end of loop on first pass

currentType

finalType

array 6
subs=
comp=

int 2

range 1 3...

in DoMakeArray
Array Definitions

integer array [one_five] [one_three] *a, b ;

currentType

finalType

array 30
subs=
comp=
range 1
5...
array 6
subs=
comp=
range 1
3...
int 2

in DoMakeArray
Array Definitions

integer array [one_five] [one_three] *a, b;

End of 2nd pass

finalType

array 30
subs=
comp=
range 1 5...

array 6
subs=
comp=
range 1 3...

int 2

type vector
type vector

type type

in DoMakeArray
Array Definitions

integer array [one_five] [one_three] *a, b;

currentType

finalType

array 30
subs=
comp=

range 1
5...

array 6
subs=
comp=

range 1
3...

int 2

in DoMakeArray

vector

type

type

type
Array Definitions

integer array [one_five] [one_three] *a, b ;

DoDeclVars exactly as before
Array Definitions

Notes:
Type pointers represent linked lists (if we have records then trees actually)

We may need to “walk” the list to process an array type.

The “head” pointer represents the type as a whole.
Subscript reduction

\[ x := a[3] \cdot [m] ; \]
Subscript reduction

\[ x := a[3] \cdot [m]; \]

<table>
<thead>
<tr>
<th>Operation</th>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>R0</td>
<td>#3</td>
</tr>
<tr>
<td>TRNG</td>
<td>R0</td>
<td>+..(R14)</td>
</tr>
<tr>
<td>IS</td>
<td>R0</td>
<td>#1</td>
</tr>
<tr>
<td>IM</td>
<td>R0</td>
<td>#6</td>
</tr>
<tr>
<td>LDA</td>
<td>R1</td>
<td>+4(R15)</td>
</tr>
<tr>
<td>IA</td>
<td>R1</td>
<td>R0</td>
</tr>
</tbody>
</table>

Expr int 3

Expr “a” (1,4,false)

Array 30

Subs=

Comp=

Array 6

Subs=

Comp=

Int 2

Range 1

Range 1

Range 3...

Range 5...
Subscript reduction

\[ x := a[3] \cdot [m] ; \]

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>R0, #3</td>
</tr>
<tr>
<td>TRNG</td>
<td>R0, +..(R14)</td>
</tr>
<tr>
<td>IC</td>
<td>R0, #1</td>
</tr>
<tr>
<td>IM</td>
<td>R0, #6</td>
</tr>
<tr>
<td>LDA</td>
<td>R1, +4(R15)</td>
</tr>
<tr>
<td>IA</td>
<td>R1, R0</td>
</tr>
</tbody>
</table>

This represents \( a[3] \)
Subscript reduction

\[ x := a[3][m] \cdot ; \]

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Registers</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>R0, #3</td>
</tr>
<tr>
<td>TRNG</td>
<td>R0, +..(R14)</td>
</tr>
<tr>
<td>IS</td>
<td>R0, #1</td>
</tr>
<tr>
<td>IM</td>
<td>R0, #6</td>
</tr>
<tr>
<td>LDA</td>
<td>R1, +4(R15)</td>
</tr>
<tr>
<td>IA</td>
<td>R1, R0</td>
</tr>
</tbody>
</table>

expr var
(0,1,true)

expr "m"
(1,6,false)

array 30
subs=
comp=
range
1
5...

array 6
subs=
comp=
range
1
3...

int 2

Subscript(second call)
Subscript reduction

\[ x := a[3][m]; \]

In Subscript(second call)

\[
\begin{align*}
LD & \quad R0, \#3 \\
TRNG & \quad R0, +..(R14) \\
IS & \quad R0, \#1 \\
IM & \quad R0, \#6 \\
LDA & \quad R1, +4(R15) \\
IA & \quad R1, R0 \\
LD & \quad R0, +6(R15) \\
TRNG & \quad R0, +..R14) \\
IS & \quad R0 \#1 \\
IM & \quad R0, \#2 \\
IA & \quad R1, R0 \\
\end{align*}
\]
Subscript reduction

LD R0, #3
TRNG R0, +..(R14)
IS R0, #1
IM R0, #6
LDA R1, +4(R15)
IA R1, R0
LD R0, +6(R15)
TRNG R0, +..R14)
IS R0 #1
IM R0, #2
IA R1, R0

return value

expr var (0,1,true)

This represents

array 30
subs=
comp=

array 6
subs=
comp=

int 2

in Subscript(second call)
**Subscript reduction**

\[
x := a[3][m] ;
\]

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Register</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>R0</td>
<td>#3</td>
</tr>
<tr>
<td>TRNG</td>
<td>R0</td>
<td>+..(R14)</td>
</tr>
<tr>
<td>IS</td>
<td>R0</td>
<td>#1</td>
</tr>
<tr>
<td>IM</td>
<td>R0</td>
<td>#6</td>
</tr>
<tr>
<td>LD</td>
<td>R1</td>
<td>+4(R15)</td>
</tr>
<tr>
<td>IA</td>
<td>R1</td>
<td>R0</td>
</tr>
<tr>
<td>LD</td>
<td>R0</td>
<td>+6(R15)</td>
</tr>
<tr>
<td>TRNG</td>
<td>R0</td>
<td>+..R14</td>
</tr>
<tr>
<td>IS</td>
<td>R0</td>
<td>#1</td>
</tr>
<tr>
<td>IM</td>
<td>R0</td>
<td>#2</td>
</tr>
<tr>
<td>IA</td>
<td>R1</td>
<td>R0</td>
</tr>
</tbody>
</table>

This represents

\[
a[3][m]
\]

Assign: exactly as before
Subscript reduction

Notes:
ReduceSubscript handles only one subscript. It is called multiple times if there are many.

It needs to check its two entries for legality--top = integer type, next has array type.
Range Checking

integer range [1..3] one_three ;

Bounds offset for this array would be +0

newType

Assuming no constants had yet been allocated in C1 block

HALT

C1

INT 1

INT 3

back in DoMakeRange
Range Checking

integer range [1..5] one_five;

Bounds offset for this array would be +

newType

range 2
lo=1
hi = 5
bo =4
base =

In actuality, you enter two constants into the constant table, lobound and highbound