Expression Evaluation

Expression Evaluation Topics

- Order of Evaluation
- Sequence Points
- Conversions
- Real Numbers

Order of Evaluation

- Not always predictable

```
/* maybe 'inx  *  jnx' is evaluated first  *
 * maybe 'jnx  *  knx' is evaluated first */
abc  =  inx  *  jnx  *  knx;
/* Adding parens doesn't help...  this is
  still unpredictable */
abc  =  (inx  * jnx)  *  knx;
```

- Unary plus can be used to introduce predictability

```
abc  =  +(inx  * jnx)  *  knx;
```

- Order of evaluation is only guaranteed between *sequence points*

Sequence Points

- A *sequence point* in an expression controls the evaluation and optimization of the expression
  - Optimizations are permitted within sequence points, but not across sequence points
  - An expression to the left of a sequence point must be fully evaluated before expressions to the right
  - Side effects resulting from evaluation of an expression to the right of a sequence point must not affect any expression to the left of a sequence point
  - Interrupts are not to allowed when an expression between two sequence points is partially evaluated

Sequence Points

- The sequence point are
  - At a function call immediately after all arguments have been evaluated
  - The end of the first operand of the operators *logical and* (&&), *logical or* (||), *conditional* (?:), and *comma* (,)
  - The end of a full expression
  - The end of an initializer
  - A controlling expression
  - The expression in a return statement

Conversions

- Conversions in non-assignment expressions
  - *floats* are always promoted to at least type *double*.
    - Given real values of different sizes, the shorter value is converted to the type of the wider value
    - Given integer and real values, the integer value is always converted to a real value

Conversions

- Conversions in non-assignment expressions
  - *shorts* are always promoted at least to type *int*
    - Given integer values of different sizes, the shorter value is converted to the type of the wider value
    - Given signed and unsigned integers, the signed value is converted to unsigned

Conversions

- Conversions in assignment expressions
  - If the source value is real, and the target is integer, the real value is truncated
  - If the source value is greater than the target can accommodate, the result is unpredictability
    - If the source and target are both integers, silent overflow *usually* occurs
    - If the source is real, a signal is *usually* generated

Conversions

- Conversions in assignment expressions
  - If …
    - the source is a real value, AND
    - the target is a smaller real value, AND
    - the source is smaller than the target can accommodate,
    - …than, an *underflow* condition exists and a signal is *usually* generated

Real Numbers

- A single real value may have more than one floating point representation.

```
/* the following code will probably    *
 * execute incorrectly                 */
double   alpha =
   pow( pow( 6, 4 ), 0.249999999999999 );
double   alef  = sqrt(  36.0  );
if  (  alpha  ==  alef  )
  puts(  "values are the same"  );
else
  puts(  "values are different"  );
```

Real Numbers

- **To correctly compare floating point values you must use the *epsilon test*.**

```
#include <math.h>
#define EPSILON            (1E-5)
#define REAL_EQ( r1, r2 ) \
               (fabs((r1) - (r2)) < EPSILON)
#define REAL_GT( r1, r2 ) \
               ((r1) - (r2) > EPSILON)
#define REAL_LT( r1, r2 ) \
               ((r1) - (r2) < EPSILON)
 . . .
if  (  REAL_EQ(  alpha, alef  )  )
   . . .
```