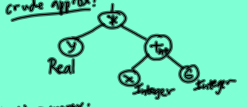
Expressions

```
var
  x : Integer;
  y,z : Real;
  :
z := y*(x+6)
```

crude approx:



better approx:

Type conversions (implicit)

1. unary conversions — conversion of a type based on the type alone (regardless of context)

   In Pascal:
   - Single → Real
   - float → double
   - Subrange → base type // no-cost machine conversion

   In C, etc.
   - char → int
   - short → int
   - float → double
   - array of T → pointer to T

unary conversions are applied first to arguments of any arith or comparison ops.

2. Binary conversion — when a binary operator is applied, first arg is unary converted by itself, then each arg is binary converted based on the type of the other arg.
   - convert Integer → Real if the other arg is Real

Real division op in Pascal (/) requires two Real args (may need to convert both args from Integer to Real).

3. Others, incl. assignment conversions (r.h.s. of := converted to type of l.h.s. before the assignment.
   - Real → Single // okay
   - Real → Integer // error
   - pointer to void → pointer to T [nil constant] (no-cost machine conversion) (same array element type)

Parameter conversions: convert actual arg to type of corresp formal param of a proc/fcn. Same as assignment conversion

---

Encode this tree by traversing post order, generating assembly code to evaluate expr at runtime

Runtime environment:
Use the control stack
Model as a stack machine:

Evaluating an expr at runtime pushes the value onto the stack (net result)

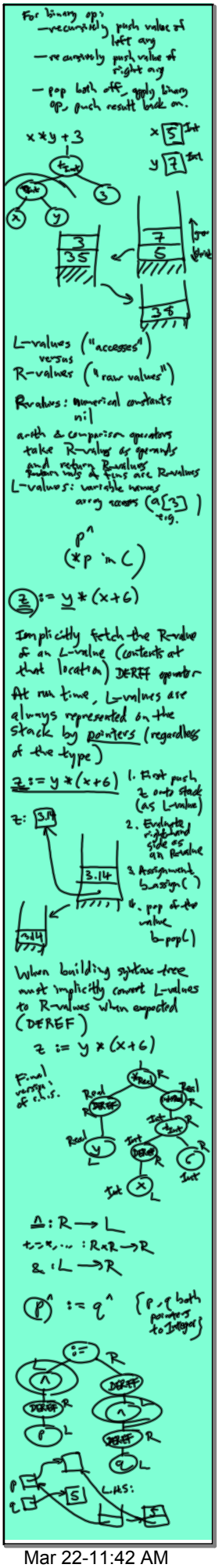
Recursively, bottom up:

For constants: push value onto stack directly

For vars: push (basically same)

For unary ops, recursively eval Subexpr (argument) value now on top of stack. Pop off, apply operator, then push result back on

Mar 22-11:03 AM


For binary op's
—recursively push value of left arg
—recursively push value of right arg
— pop both off, apply binary op, push result back on.
x*y+3
x 5 Int
y 7 Int
L-values ("accesses")
versus
R-values ("raw values")
R-values: numerical constants
nil
arith & comparison operators take R-values as operands and return R-values
assignments only & fns are R-values
L-values: variable names
array access (a[3] e.g.)
p^
(*p in C)
z := y * (x+6)
Implicitly fetch the R-value of an L-value (contents at that location) DEREF operator
At run time, L-values are always represented on the stack by pointers (regardless of the type)
z := y * (x+6)
1. First push z onto stack (as L-value)
2. Evaluate right-hand side as an R-value
3 Assignment, b_assign()
4. pop off the value b_pop()
z: 3.14
3.14
3.14
When building syntax tree must implicitly convert L-values to R-values when expected (DEREF)
z := y * (x+6)
Final version of t.s.
Real
Int
:= : R → L
+ - * ... : R x R → R
& : L → R
p^ := q^ {p, q both pointers to Integer}
LHS:
5