CSE 3302 Notes 7: Expressions & Assignment

(Last updated 7/15/14 7:17 PM)

7.2. Arithmetic Expressions

Review items for infix expressions:

Precedence (C/C++/Java/JavaScript vs. Pascal)
Associativity
Arity - binary, unary, ternary (`?` `:`)
Dijkstra's shunting yard (http://en.wikipedia.org/wiki/Shunting-yard_algorithm)

C function call arguments are not required to be processed in a particular order (but is usually right-to-left, see http://ranger.uta.edu/~weems/NOTES3302/NEWNOTES/NOTES07/argOrder.c)

C etc. comma operator evaluates left-to-right (uses right operand value as result)

7.3. Overloaded Operations

C++ overloaded operators for iterators - (appendix A of
http://graphics.stanford.edu/courses/cs368-00-spring/manuals/CGAL_Tutorial.pdf)

(Nice examples in chapter 11 of Stroustrup, *The C++ Programming Language*)

7.4. Conversions

(type)

Narrowing - going from a value in a large set to a small set (often unsafe, can't "undo" back to original values)
Widening - going from a value in a small set to a large set (often safe, can "undo" to original value)
Coercion - implicit conversion, defined by language/implementation (*mixed mode* operations)
Conversion - explicit "cast" by programmer

(http://ranger.uta.edu/~weems/NOTES3302/NEWNOTES/NOTES07/conversion.c)

JavaScript - `==` vs. `===` and `!=` vs. `!==`

7.5. Relational and Boolean Expressions

Fundamental difficulties with equality in logic & mathematics . . .

Notions of *equivalence* may be defined WRT a single function

Is an integer odd or even?
Is a function g in $\Theta(f)$?

What about *equality*?

Has to cover *all* notions of equivalence (addresses and references?)

For *x* and *y* to be equal, they are indistinguishable to any function

PLs

Shallow equality test - no dereferencing, tests whether values refer to same object?

Deep equality test - dereference and check values (cycles . . .)

ML = deep for *equality* types

Doesn't include `real`

```
- (1,2,3)=(1,2,3);
val it = true : bool

- [1,2,3]=[1,2,3];
val it = true : bool

- beatles=beatles;
val it = true : bool

- beatles=tl(beatles);
val it = false : bool

- [hd(beatles)]=tl(beatles);
val it = false : bool
```

(ML does allow `ref` types which function like pointers)

Scheme

`eq?` (shallow) and `equal?` (deep)

C

Besides comparing pointers with `==` and `!=`, can also use other comparisons (meaningful when dealing within same array, `struct`, etc.)

Pascal

Pointers may only be compared using equality comparisons (`=`, `<>`)

Lead-ins to next section:

Boolean operators to force sub-expression evaluation (for side effects)

C - Use `&` or `*` in place of `&&`, | or `+` in place of | |

JavaScript `undefined`

Used when a property does not exist for an object.

To access `a.b.c.d` or get `undefined` (to avoid `TypeError`):

```
dCheck = a && a.b && a.b.c && a.b.c.d;
```

Based on short-circuit evaluation, JavaScript uses the last truthy/falsy value as result for `&&` and | | (so do Scheme `and`/`or`, but `0` is truthy and only `#f` is falsy).

Misspelled property name vs. property with `undefined` as value . . .

`!!` sanitizes truthy/falsy value to `true` or `false`
`a || b` in JavaScript may be achieved in C using `a ? a : b`
`a && b` in JavaScript may be achieved in C using `a ? b : a`

7.6. Short-Circuit Boolean Evaluation

C:

Left side of | | and `&&` is determined before right side, i.e. no portion of right side is evaluated before left side is determined.

"Give equivalent C code (e.g. using `if ... else ...`) to demonstrate the short-circuit nature of C boolean operators. Do not use `&&`, | |, or `!` in your solution! Do not use work variables!"

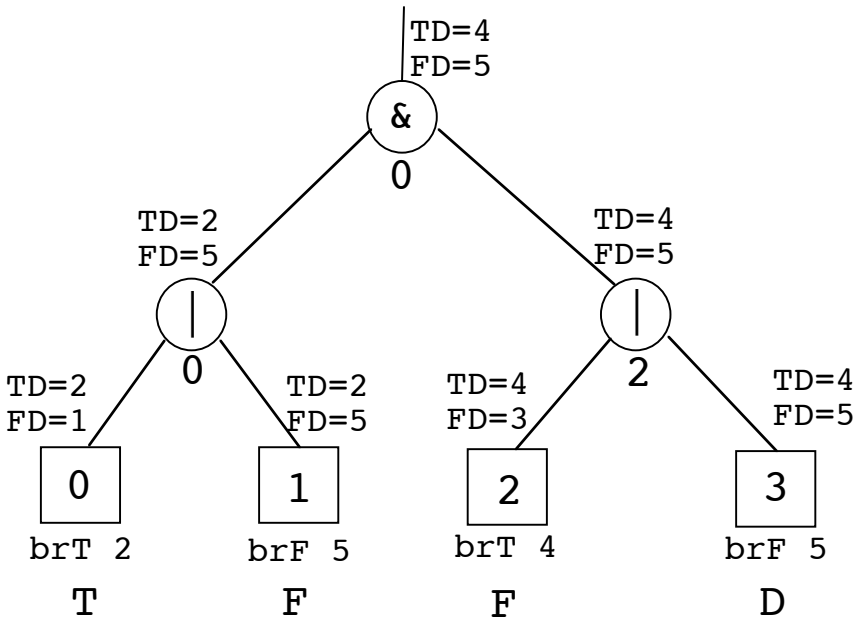
```
result = a < 13 && a > 10;
```

```
if (a < 13)
  if (a > 10)
    result = 1;
  else
    result = 0;
else
  result = 0;
```

```
result = e < 25 && !(f > 55 && g < 66);
```

```
if (e < 25)
  if (f <= 55)
    result = 1;
  else if (g >= 66)
    result = 1;
  else
    result = 0;
else
  result = 0;
```

http://ranger.uta.edu/~weems/NOTES3302/lab3fall12.pdf - Conversion of expression with boolean result to jump-based code



7.7. Assignment

Shallow and *deep* differences apply again

Multiway (simultaneous, parallel) assignment

```
a, b = b, a;

i, j, a[i], a[j] = j, a[i], a[j], i;
```

What does this really save?
JavaScript 1.7 - Destructuring assignment (also common in SML, but strongly typed)

```
[a,b] = [1,2];
[a,b] = [b+1,a+3];
[a,a] = [b+2,a+1];
```
What happens?

7.8. Mixed-Mode Assignments

No different than coercion - other than applying to lvalue rather than rvalue