3.3 Conditional Evaluation

- Want to support if/then/else
- We'll represent false as 0, true as 1:

```
--> if 1 then 2 else 3
2
```

- Syntax (BNF):

<expression> ::= **if** <expression> **then** <expression>

else <expression>

```
if (test-exp true-exp false-exp)
```

3.3 Conditional Evaluation: Step 1

Add some code to the grammar:

```
((define grammar-3-3
  '((program (expression) a-program)
    (expression (number) lit-exp)
    (expression (identifier) var-exp)
    . . .
    (expression
      ("if" expression
       "then" expression
       "else" expression) if-exp)
```

3.3 Conditional Evaluation: Step 2

Add some code to datatype definition:

```
(define-datatype expression expression?
  ...
  (if-exp
    (test-exp  expression?)
    (true-exp  expression?)
    (false-exp  expression?))
```

3.3 Conditional Evaluation: Step 3

Add some code to **`eval-expression`**:

```
(define eval-expression
  (lambda (exp env)
    (cases expression exp
      ...
      (if-exp (test-exp true-exp false-exp)
        (if (true-value?
                (eval-expression
                    test-exp env))
            (eval-expression true-exp env)
            (eval-expression false-exp env)))
```

3.3 Conditional Evaluation

```
; hardware-dependent?
(define true-value?
  (lambda (x)
    (not (zero? x))))
```