CSSE 304 Day 15 Summary

1. What are the three main ingredients of an abstract datatype?
   a.
   b.
   c.

2. Specification and possible representations/implementations of "non-negative integers" ADT.
   a. The four base (defining) operations (procedures) :
      i. zero iszero? succ pred (may be undefined for input zero)

   b. Sample derived operation: `add`. Write it in terms of the base operations:

   c. Implementation 1: Unary. ⌈0⌉ = '( ) ⌈n+1⌉ = (cons #t ⌈n⌉). How to define operations in Scheme? Other implementations are in the online slides. Due to time constraints, will not do them in class.

3. Aggregate datatypes:
   a. Arrays

   b. Records

   c. Union types (variant records):

4. `define-datatype`. A way to define new (possibly recursive) "record" types with type-checking for the fields.
   a. **`define-datatype`** creates constructors for immutable variant records. In order to use it, you must first load `chez-init.ss`.

   b. Constructors check the types of the fields, and report an error if incorrect.

   c. **`cases`** is used to get references to the various fields.

   d. Details of syntax for defining and using datatypes are in the slides.

   e. We examine datatypes for binary trees, s-lists, lambda-calculus expressions. A place for your notes:

f.
```
(define inorder
  (lambda (tree)
    (cases bintree tree
```

5. Code is data. In Scheme, both have the same form. `eval` treats code as data. Don't use it in your interpreter project code!

6. A datatype for lambda-calculus expressions (you will extend this definition to include other Scheme syntax).

```
(define-datatype expression expression?
  [var-exp
    (id symbol?)]
```

```
(define-datatype expression expression?
  [var-exp
    (id symbol?)]
  [lambda-exp
    (id symbol?)
    (body expression?)]
  [app-exp
    (rator expression?)
    (rand expression?)])
```

7. How will the `app-exp` variant of the `expression` datatype change if we allow any number of arguments in a procedure application?

   How will the `app-exp` variant of the `expression` datatype change if we allow any number of arguments in a procedure application?

8. Add a new variant to the `expression` type definition so we can include `set!` expressions.

9. `parse-exp` transforms "raw" code (concrete syntax) into an Abstract Syntax Tree (AST).

10. One of the slides has the code for parsing basic lambda-calculus expressions. You will enhance it in HW 11b. Study it and come to the next class prepared to ask questions on anything you do not understand.

11. Add a case to `parse-exp` that parses a `set!` expression, producing an instance of the variant that you defined in question 6.

12. Two of the slides describe how you must handle errors in A11b so the PLC server will give you credit. `(eopl:error 'parse-exp "bad let*: ~s" exp)`

13. `unparse-exp` is an example of one way we can use a parsed expression. So is `occurs-free`.