CS 230
Programming Languages

10 / 24 / 2022

Instructor: Michael Eckmann

Today's Topics

- Questions? / Comments?
- Finish discussion of Enumerations
- Start Functional Languages

Ordinal / Enumeration

- Ordinal types
  - Range of values are associated with positive integers
- Many languages provide support for user defined ordinal types like Enumerations.
- Enumeration types
  - Named constants represent positive integers
  - Design issues
    - Are enumeration type values coerced to integer?
      - Same operations and range of values of integers
    - Are any types coerced to enumeration type values?
      - Could break the allowable values of the enum type
  - Range of values intentionally limited.

Enumeration

- e.g. In C#

```
enum months {Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec};
```

- e.g. In C++

```
enum months {Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec};


 months m;        // declare a variable of type months
```

- Represented as 0 to 11 typically but could be given programmer specified values in the declaration.
- They look the same, but C# enum types not coerced to integer whereas C++'s are --- so, C# doesn't allow operations that may not make sense, while C++ does. e.g. In C++, we can add 1 to an enum

Enumeration

- e.g. In C++

```
enum months {Jan, Feb, Mar, Apr, May, Jun, Jul,
Aug, Sep, Oct, Nov, Dec};


// declares currentMonth of type months and assigns
// an initial value
months currentMonth = Jun;
```

- In C++, we can add 1 to an enum
  - e.g. currentMonth++;
- Any danger here?

Enumeration

- Java provides support of enumerations (since Java 1.5). When used, they create full classes for the enum. See:

http://java.sun.com/j2se/1.5.0/docs/guide/language/enums.html

- What are advantages to enumeration types?

Enumeration

- What are advantages to enumeration types?
  - All forms of implementation of them offer better readability
  - Reliability
    - No arithmetic operations on the enum types in Ada and C# --- this prevents adding/subtracting which might not make sense like in adding months together
    - Ada and C# also restrict values within the range for the type.
    - C treats them like integers so we don't get those advantages

Enumeration

- Reliability
  - C++ allows numeric values to be assigned to enum types if they are explicitly cast.
  - C++ also allows values to be assigned to enum types but the range is checked for validity.
    - This just checks that the integer is between the lowest and highest enum value. Enum values need not be consecutive integers. Any problem here?

Functional Programming Languages

- Based on mathematical functions
- They map a domain to a range.
- A few things about Mathematical Functions.
- The first functional language was LISP (created by John McCarthy)

Scheme

- We'll now turn our attention to Scheme and focus on its non-imperative features. Note: If Scheme were purely functional, then it would have no imperative features.
- Scheme, created in the mid-1970's, is a dialect of LISP.

Scheme

- Scheme
  - Static scoping exclusively
  - Small size
  - Functions are first-class citizens
    - Can be values of expressions
    - Can be elements of lists
    - Can be passed as parameters
  - simple syntax --- syntax is consistent
  - simple semantics

Scheme

- Scheme
  - Expressions are evaluated by the function EVAL
  - Literals evaluate to themselves
  - Function calls are evaluated by
    - First evaluate all the parameter expressions
    - Then evaluate the function after the values of the parameters are known
    - The value of the last expression in the body is the value of the function
    - All but the last should be familiar to imperative programmers.

Scheme

- Scheme
  - Primitive functions

Arithmetic: **`+, -, *, /, ABS, SQRT, REMAINDER, MIN, MAX`**

e.g., (+ 5 2) yields 7

what would (- 24 (* 5 3)) yield?

- If * is given no parameters, it returns 1 (multiplicative identity.)
- If + is given no parameters, it returns 0 (additive identity.)

Scheme

- Scheme
  - Primitive functions
    - If – is given more than two parameters, it acts as if the second through the last are summed and this sum is subtracted from the first.
    - If / is given more than two parameters, it acts as if the second through the last are multiplied together and this product is divided into the first.

Scheme

- Scheme
  - Primitive functions

QUOTE -takes one parameter; returns the parameter without evaluation

- **QUOTE** is required because the Scheme interpreter, named **EVAL**, always evaluates parameters to function applications before applying the function. **QUOTE** is used to avoid parameter evaluation when it is not appropriate
- **QUOTE** can be abbreviated with the apostrophe prefix operator

e.g., '(A B) is equivalent to (**QUOTE** (A B))

Scheme

- Scheme
  - If you wanted a list with the symbols / 8 4 in that order and you did this:

(/ 8 4)

    - Scheme would evaluate that as a function and the result will be 2
  - So, to have the list not be evaluated, use the QUOTE:

'(/ 8 4)

    - Scheme gives us the list with those three elements; no evaluation occurs.
    - alternatively (QUOTE (/ 8 4))

Scheme

- DEFINE
  - Binds a name to a value
    - Different than a variable --- it is a named constant.
  - Binds a name to a lambda expression

Scheme

- Scheme
  - (DEFINE hello 5.6)
  - If you wanted the symbol hello instead of the value in the constant named hello

'hello

vs.

hello

  - The first one gives us hello
  - The second one evaluates to 5.6

Scheme

Functions on lists

- CAR
- CDR
- CONS
- LIST

Scheme

`CAR` takes a list parameter; returns the first **element** of that list

e.g., (CAR '(A B C)) yields A

(CAR '((A B) C D)) yields (A B)

Notice: CAR can evaluate to an **atom** or a **list**.

`CDR` takes a list parameter; returns the **list** after removing its first element

e.g., (CDR '(A B C)) yields (B C)

(CDR '((A B) C D)) yields (C D)

(CDR '(A B)) yields (B)

(CDR '(A)) yields ()

Notice: CDR always evaluates to a **list**.