Introduction to Problem Solving and Abstraction
Lecture on September 25, 2019

Topics to cover: Lab post-mortem, review of special forms needed in lab: if, when, cond, and, or; Start of chapter 12

Overview of lab

;; PRINTF
;; Usage: (printf format-string expr1 ... exprn).
;;
;; The printf function causes the FORMAT-STRING (i.e., its first argument) to be
;; displayed in the Interactions Window, except that:
;; 1. each instance of the escape sequences, \n or \%, in the format-string
;; are interpreted as new-line characters, and thus cause new-lines in
;; the Interactions Window;
;; 2. each instance of the escape sequences, ~A or ~a in the format-string
;; are replaced by character sequences representing the values of the
;; corresponding input expressions; and
;; 3. if the format-string contains n instances of ~A (or ~a), then there
;; must be n expression arguments following format-string, as follows:
;; (printf format-string expr_1 ...expr_n).
;;

In lab 3, the functions you wrote all involved decision making. For some of the functions, the solutions required both the use of relational and logical operators. Relational operators exist for many data types, including =, <, >, <=, >= for numbers. For booleans, quoted symbols, and numbers, the eq? operator can be used for checking equality.

For some of the functions, you chose to use nested if statements and for others, the cond statement. In general, the cond expression is preferred over nested if statements for readability and compactness of function.

The eq? function is used when comparing booleans, numbers, and quoted symbols. Remember, this function is just eq?, not eq=? . There is no =? operator.

See problems 7.1 and 7.2 in our book: Summarize the following information about the define and quote special forms:

(1) how many elements there are–aside from the keyword symbol;
(2) which elements get evaluated and which do not;
(3) whether there is an output value and, if so, how it is computed; and
whether there is a side effect and, if so, what it is.

For each statement below, decide which of the words in parentheses apply and specify the side effect generated, if any:

- Evaluation of a define special form (always, never, sometimes) causes a side effect.
- Evaluation of a quote special form (always, never, sometimes) causes a side effect.
- Evaluation of a printf function (always, never, sometimes) causes a side effect.

Look into inserted multiple statements in a function body. For example, put some printf statements into the letter-grade function.

When interpreting \textit{predicateExpr}, anything that does not evaluate explicitly to \#f evaluates to \#t. Here are examples of the “non-strict” version of the if statement:

- (if “hi” 8 \#f) \implies 8
- (if 42 "The answer to life, the universe, and everything" "What’s the question?"
  \implies “The answer to life, the universe, and everything"
- (if (not (/ 10 2)) 'happy 'joy) \implies joy

Many special forms use \textit{lazy} evaluation, meaning all parts of the special form are never evaluated, contrary to the DER:

- \texttt{if}: the true part or the false part is evaluated, never both.
- \texttt{when}: the body is evaluated only if the condition is true.
- \texttt{cond}: only one clause body is ever evaluated.
- \texttt{and}: the boolean expressions (strict or non-strict) are evaluated until a \#f occurs.
- \texttt{or}: the boolean expressions (strict or non-strict) are evaluated until a \#t occurs.

For each of these \textit{lazy special forms}, you should be able to distinguish the parts that are and are not evaluated in each expression call.

Talk about the \texttt{when} special form and how it can be used as an else-less expression.

Talk about using side-effect printing in a decision statement and which statements you can use it in.
Chapter 13: Conditional Expressions II

1. Nested IF statements:
   If there are more than two answers to choose from, you would like to have an \textit{n-ary} special form that makes \textit{multi-way} decisions. This can be done by using \textit{nested if} expressions. For example, consider the function \texttt{letter-grade} with the following contract and definition:

   \begin{verbatim}
   ;; Name: letter-grade
   ;;---------------------------------------
   ;; Usage: (letter-grade score) --> quoted symbol
   ;; Input: score is a number between 0 and 100
   ;; Output: One of the quoted symbols A, B, C, D

   (define letter-grade
      (lambda (score)
         (if (>= score 90)  ;; true part of 1st if
             'A ;; true part of 1st if
             (if (>= score 80) ;; false part of 1st if
                 'B ;; true part of 2nd if
                 (if (>= score 70) ;; false part of 2nd if
                     'C ;; true part of 3rd if
                     'D ) ;; false part of 3rd if
                 ) ;; end of 2nd if
             ) ;; end of 1st if
         ) ;; end of lambda
      ) ;; end of define
   
   You will find another special form in Chapter 13, called \texttt{cond}, that is most commonly used for making multi-way or \textit{n-ary} decisions.

2. The COND special form

   \begin{verbatim}
   ;; Name: letter-grade-v2
   ;;---------------------------------------
   ;; Usage: (letter-grade-v2 score) --> quoted symbol
   ;; Input: score is a number between 0 and 100
   ;; Output: One of the quoted symbols A, B, C, D

   (define letter-grade
      (lambda (score)
         (cond
            (( >= score 90)  ;; case 1: got an A
             'A
            )
            (( >= score 80) ;; case 2: got a B
             'B
            )
            ((>= score 70) ;; case 3: got a C
             'C
            )
            (else
             'D ) ;; case 4: got a D
         )
      )
   
   You will find another special form in Chapter 13, called \texttt{cond}, that is most commonly used for making multi-way or \textit{n-ary} decisions.

   \end{verbatim}
Demonstrate side-effect printing in the function above. Mention that side-effect printing is not easily done in an if special form.

Two more special forms are the logical operators AND and OR, and the function NOT. Each of these special forms returns a boolean.

3. The AND and OR special forms

The syntax of the AND special form is as follows:

\[(\text{and } boolean_1 \ boolean_2 \ \ldots \ boolean_n) \rightarrow \text{boolean}\]

The and special form takes in any number of boolean expressions and returns #t only if all the boolean_i expressions evaluate to #t. Evaluation of the and special form stops when the first boolean_i expression evaluates to #f. This is known as short-circuit or lazy evaluation and it is this feature that makes the and special form different from the DER.

Write truth tables for the and and or special forms (see page

The syntax of the OR special form is as follows:

\[(\text{or } boolean_1 \ boolean_2 \ \ldots \ boolean_n)\]

The or special form takes in any number of boolean expressions and returns #f only if all the boolean_i expressions evaluate to #f. Evaluation of the or special form stops when the first boolean_i expression evaluates to #t. This is known as short-circuit or lazy evaluation and it is this feature that makes the or special form different from the DER.

You should be able to look at the usage of the and and or special forms and determine which parts are and which parts are not evaluated.

4. The NOT operator

If you look not up in the help desk of DrR, you’ll find that it is a function that consumes one boolean and returns the opposite of that boolean’s truth value.
Remember, anything that is not explicitly #f is true. When you use anything but #t to represent a true return, you are using the “non-strict” form of a boolean.

How could we write the not function if one was not built-in?

If you use an expression such as (if (> x y) #t #f), you could do the same by just evaluating (> x y). You should never have to use a literal truth value as a return.

(eq? boolie #t) ==> boolie
(if (eq? boolie #t) thenExpr elseExpr) ==> (if boolie thenExpr elseExpr)