Worksheet #3: Michael Collins

Write a procedure that computes the product of two non-negative integers, using only the operation of addition, and simple tests.

Hint: x*y for positive x, y can be defined as

\[ \begin{align*}
0 & \quad \text{if } x = 0, \\
y + (x-1)y & \quad \text{otherwise}
\end{align*} \]

ANSWER:

\[
(define \text{mult} \ x \ y)
\]
\[
\begin{align*}
  & (\text{if} \ (\equiv \ x \ 0) \\
  & \quad 0 \ \\
  & \quad (+ \ y \ \text{mult} \ (- \ 1) \ y)))
\end{align*}
\]

Exponentiation: Define a procedure that given as input integers b and n, calculates b^n (b^n = b raised to the power n), using simple tests and the multiply operator ‘*’.

Hint: b^n can be defined as:

\[ \begin{align*}
1 & \quad \text{if } n = 0, \\
b \ast b^{(n-1)} & \quad \text{otherwise}
\end{align*} \]

ANSWER:

\[
(define \text{expt} \ b \ n)
\]
\[
\begin{align*}
  & (\text{if} \ (\equiv \ n \ 1) \\
  & \quad b \ \\
  & \quad (* \ b \ \text{expt} \ b \ (- \ 1)))
\end{align*}
\]

How many multiply operations are required to calculate 5^1000?

Exponentiation 2: Can you think of another procedure definition for exponentiation which takes fewer multiply operations?

Hint: We can write

\[ \begin{align*}
b^n & \quad = \ 1 \quad \text{if } (n = 0) \\
& \quad = (b^{(n/2)})^2 \quad \text{if } (n \text{ is even}) \\
& \quad = b \ast (b^{(n-1)}) \quad \text{if } (n \text{ is odd})
\end{align*} \]

Assume we have procedures
\quad (even? \ x) \ which \ returns \ true \ if \ x \ is \ even;
(square x) which calculates x*x = x^2

ANSWER:

(define (expt b n)
  (cond
    ((= n 0) 1)
    ((even? n) (square (expt b (/ n 2))))
    (else (* b (expt b (- n 1))))))
Write a procedure that computes the Fibonacci numbers:
the n’th number fib(n) is defined as

\[
\begin{align*}
\text{fib}(n) &= 0 \quad \text{if } (n=0) \\
&= 1 \quad \text{if } (n=1) \\
&= \text{fib}(n-1) + \text{fib}(n-2) \quad \text{otherwise}
\end{align*}
\]

ANSWER: (but it’s very inefficient!!)

\[
\begin{align*}
\text{(define (fib n)} \\
&\quad \text{(cond}} \\
&\quad \quad (\text{((= n 0) 0))} \\
&\quad \quad (\text{((= n 1) 1)} \\
&\quad \quad (\text{else (+ (fib (- n 1)) (fib (- n 2)))))})
\end{align*}
\]

Recall the recursive procedure for calculating the sum of the integers between x and y inclusive:

\[
\begin{align*}
\text{(define sum (lambda (x y)} \\
&\quad (\text{(if (> x y)}} \\
&\quad \quad 0 \\
&\quad \quad (\text{(+ x (sum (+ x 1) y))))))
\end{align*}
\]

Now write a procedure that computes the sum of the integers between x and y inclusive, but where the process generated by the procedure should be *iterative*.

ANSWER:

\[
\begin{align*}
\text{(define (sum-iter i sum y)} \\
&\quad (\text{(if (> i y)}} \\
&\quad \quad \text{sum} \\
&\quad \quad \text{(sum-iter (+ i 1) (+ sum i) y))})
\end{align*}
\]

\[
\begin{align*}
\text{(define (sum x y) (sum-iter x 0 y))}
\end{align*}
\]

Write a procedure that computes the product of two non-negative integers, using only the operation of addition, and simple tests. The process generated by the procedure should be *iterative*.
\[\text{ANSWER:}\]

\[
\begin{align*}
\text{(define (mult-iter i sum x y)} & \\
& \quad \text{(if (= i x) sum)} \\
& \quad \text{(mult-iter (+ i 1) (+ sum y) x y)))}
\end{align*}
\]

\[
\text{(define (mult x y) (mult-iter 0 0 x y))}
\]

\[\begin{align*}
\text{Write a procedure that computes the Fibonacci numbers, but whose process is iterative}\end{align*}\]

\[\text{ANSWER: (much more efficient...)}\]

\[
\begin{align*}
\text{(define (fib-iter i a b n)} & \\
& \quad \text{(if (= i n) b)} \\
& \quad \text{(fib-iter (+ i 1) (+ a b) a n)))}
\end{align*}
\]

\[
\text{(define (fib n) (fib-iter 0 1 0 n))}
\]