**CSSE 304 Assignment #12 Updated for Spring, 2014**

You may work with another student on this assignment if you wish, and I encourage you to do so. If you want to find someone to work with but don't know who to ask, send me an email message by noon on Monday, and I will give you the email addresses of others who have done the same. When you do the submission that you want to count for your grade, enter your partner's username (if you have a partner) in the text box; if you work alone, leave the text box blank.

Problems 1-3 deal with unusual representations of integers.

**#1** (40 points). bigits. Do EOPL exercise 2.1, page 34. For all of your procedures, assume that a global variable BASE has been defined to be an integer whose value is at least 2. Define the four basic integer procedures from page 32, using the bignum representation described on page 34. You can (and should) use the code for plus (page 33) as given in the book, and then use that as a helper procedure when you write similar representation-independent versions of multiply and factorial. The values returned by all of these procedures will be bignum representations, but your code for plus, multiply and factorial should not directly refer to the representation. Include all of your function definitions in your submission. To make debugging/testing easier, you must also include two procedures int->bignum and bignum->int that do the conversions based on the current value of BASE. You can choose whether the code for these procedures should be representation-independent (for efficiency purposes, I chose not to make them representation-independent in my implementation). You should not call either of these in your implementation of the bignum arithmetic procedures.

The following transcript should clarify the above requirements and perhaps reveal additional requirements:

> (begin (define BASE 2) (zero))

()

> (begin (define BASE 2) (succ (zero)))

(1)

> (begin (define BASE 2) (succ (succ (zero))))

(0 1)

> (begin (define BASE 2) (pred (succ (succ (zero)))))

(1)

> (begin (define BASE 2) (pred (pred (succ (succ (zero))))))

()

> (begin (define BASE 8) (succ '(7 0)))

(0 1)

> (begin (define BASE 8) (int->bignum 257))

(1 0 4)

> (begin (define BASE 8) (int->bignum 255))

(7 7 3)

> (begin (define BASE 8) (bignum->int (int->bignum 321654987)))

321654987

> (begin (define BASE 7) (succ (succ(int->bignum 99999))))

(6 5 3 4 6 5)

> (begin (define BASE 7) (bignum->int (succ (succ(int->bignum 99999)))))

100001

> (begin (define BASE 6) (plus (int->bignum 200) (int->bignum 17)))

(1 0 0 1)

> (begin (define BASE 6) (plus (int->bignum 1) (int->bignum 1)))

(2)

> (begin (define BASE 8) (multiply (int->bignum 2) (int->bignum 0)))

()

> (begin (define BASE 8) (multiply (int->bignum 0) (int->bignum 2)))

()

> (begin (define BASE 8) (multiply (int->bignum 12) (int->bignum 1)))

(4 1)

> (begin (define BASE 8) (multiply (int->bignum 7) (int->bignum 73)))

(7 7 7)

> (begin (define BASE 8) (multiply (succ (int->bignum 512)) (pred (int->bignum 512))))

(7 7 7 7 7 7)

> (begin (define BASE 10) (factorial '()))

(1)

> (begin (define BASE 3) (factorial '(1 1)))

(0 2 2)

> (begin (define BASE 10) (factorial '(8)))

(0 2 3 0 4)

> (begin (define BASE 2) (factorial (int->bignum 6)))

(0 0 0 0 1 0 1 1 0 1)

> (begin (define BASE 2) (bignum->int (factorial (int->bignum 6))))

720

**#2** (20 points). **This analysis portion** should be done separately and **submitted to a Moodle drop box before midnight on the day of Session 21.**  First, analyze the running time of your bignum factorial algorithm as a function of N (the number whose factorial you are computing). Explain how you get your answer. Then compare your calculation to results obtained by actually timing the procedure. Use *Chez* Scheme's time syntactic form (I demonstrated it in the rotate efficiency example in class). Time the calculations of the factorials of several numbers. Show a transcript of your calculations (and an attempt at fitting the data to your predicted curve) in your submission.

Next, analyze the effect of the BASE on the running time, then time the calculations of the same factorial with different bases to see how well it fits your analysis (you decide what value(s) of N is/are best to use. Again try to fit the results to an appropriate curve.

Be sure that your drop box submission includes the names of all group members who were substantially involved in your analysis.

**#3** (30 points) Use difference trees to implement integers, as described in EoPL exercise 2.3, page 34.  
While this is clearly not an efficient implementation, it gives you another opportunity to deal with the notion of abstract data types and their implementations. Notice that diff-trees are immutable, so it is okay for two diff-trees to share a common subtree. Define the following procedures, plus any others that you need as helpers.

(dt? obj) Is this object a diff-tree?  
(dt-negate dt) Produces a diff-tree that represents the negative of the integer   
 represented by *dt*.  
(dt+ dt1 dt2) Produces a diff-tree that represents the sum of the integers

represented by *dt1* and *dt2*.  
(dt- dt1 dt2) Produces a diff-tree that represents the integer represented by *dt1*

minus the integer represented by *dt2*.

(dt= dt1 dt2) Do diff-trees *dt1* and *dt2* represent the same integer?

The following user-interface procedures are also required, because they may be helpful as you develop test-cases and debug your code:

(dt->integer dt) Produces the integer represented by *dt*.  
(integer->dt n) Produces a diff-tree that represents the integer *n*.

**Examples:** (Note that because there are an infinite number of possible diff-trees that represent each integer, some of your answers may not be the same as mine):

> **(define one '(one))**

> **(define zero '(diff (one) (one)))**

> **(define two (dt+ one one))**

> **two**

(diff (one) (diff (diff (one) (one)) (one)))

> **(define minus-two (dt- zero two))**

> **minus-two**

(diff

(diff (one) (one))

(diff (one) (diff (diff (one) (one)) (one))))

> **(define four (dt- two minus-two))**

> **four**

(diff

(diff (one) (diff (diff (one) (one)) (one)))

(diff

(diff (one) (one))

(diff (one) (diff (diff (one) (one)) (one)))))

> **(define minus-three (dt+ one (dt-negate four)))**

> **minus-three**

(diff

(one)

(diff

(diff (one) (diff (diff (one) (one)) (one)))

(diff

(diff (one) (one))

(diff (one) (diff (diff (one) (one)) (one))))))

> **(diff-tree->integer minus-three)**

-3

> **(integer->diff-tree 8)**

(diff

(diff

(diff

(diff

(diff

(diff

(diff (one) (diff (diff (one) (one)) (one)))

(diff (diff (one) (one)) (one)))

(diff (diff (one) (one)) (one)))

(diff (diff (one) (one)) (one)))

(diff (diff (one) (one)) (one)))

(diff (diff (one) (one)) (one)))

(diff (diff (one) (one)) (one)))

**Practice with define-datatype**

**#4.** (10 points) bintree-to-list. EOPL Exercise 2.24, page 50. This is a simple introduction to using cases and the bintree datatype (bintree definition is given on page 50). See notes below on using define-datatype and bintree.

**#5.** (30 points) max-interior. EOPL Exercise 2.25, page 50. The algorithm will be the same as before, but you will write it so that it expects its input to be an object from the bintree datatype. As before, you may not use mutation. You may not traverse any subtree twice (such as by calling leaf-sum on each interior node). You may not create an additional data structure that you then traverse to get the answer. Think about how to return enough info from each recursive call to do this without another traversal.

**Code to use for #4 and #5:** Copy this code to the beginning of your file, or get it from <http://www.rose-hulman.edu/class/csse/csse304/201430/Homework/Assignment_12/12.ss>

; Binary trees using define-datatype

(load "chez-init.ss")

;; from EOPL3, page 50

(define-datatype bintree bintree?

(leaf-node

(num integer?))

(interior-node

(key symbol?)

(left-tree bintree?)

(right-tree bintree?)))

**TO USE DEFINE-DATATYPE with petite Chez Scheme on your computer:**

The chez-init.ss file should be in the same folder as your code. You can get it from:

<http://www.rose-hulman.edu/class/csse/csse304/201430/Homework/Assignment_12/chez-init.ss>

**TO USE DEFINE-DATATYPE with the Grading Server**

The chez-init.ss file is automatically loaded by the server, so you should not have   
to do anything special. Just load your own code.