

**CIS-280**  
**Project— Part I: 100 points**  
**Floating Point Arithmetic**  
**Due Thursday, April 7, 2005**

Computers use floating point arithmetic for representing real numbers; this enables them to handle very large and very small numbers and to represent each of these numbers with the same precision — ie., with the same number of significant figures. A floating point number consists of two components: a mantissa and an exponent. The mantissa, a signed decimal number greater than or equal to .1 and less than 1.0, gives the significant digits of the number; the exponent, a signed integer, specifies the power of 10 by which the mantissa must be multiplied to obtain the value of the number. The following table provides several examples; in each case, the floating point numbers are represented with five significant digits:

<b>Number</b>	<b>Mantissa</b>	<b>Exponent</b>
345.64	.34564	3
5378	.53780	4
.00326	.32600	-2

To add two floating point numbers, one must move the decimal point in one of the mantissas so that the numbers have equal exponents, add the mantissas, and then normalize the result so that the resultant mantissa and exponent conform to the rules for floating point numbers. To normalize the result, one shifts the decimal point in the resultant mantissa so that it is immediately to the left of the leftmost non-zero digit, adjusting the exponent as necessary to account for the new position of the decimal point. For example, consider adding  $.47532 \times 10^4$  and  $.35622 \times 10^2$ . To get equal exponents in the two numbers, we shift the decimal point to the left in the second number (the one with the smaller exponent), so that we are adding  $.47532 \times 10^4$  and  $.00356 \times 10^4$ . The result is  $.47888 \times 10^4$ . As a second example, consider summing  $.43564 \times 10^4$  and  $.82222 \times 10^4$ . The result is  $1.25786 \times 10^4$ , but this must be normalized to  $.12579 \times 10^5$ . (**Notice the rounding.**) Floating point numbers are subtracted in an analogous manner. Two floating point numbers are equal if and only if their mantissas and exponents are equal.

The assignment on the next page involves the construction of a floating point arithmetic package using data abstraction. You may find the following Scheme procedures useful:

- (round x)** Takes a decimal number as argument and returns it rounded to an integer
  
- (truncate x)** Takes a decimal number as argument and returns it with its fractional part dropped — ie., returns the largest integer less than or equal to x.

Consider the problem of representing and operating on non-negative floating point numbers (this means that mantissas will never be negative) with five digit mantissas. Assume that you have a constructor and two selectors:

- (make-normal-float *d e*): given a decimal number *d* where  $d=0$  or  $.1 \leq |d| < 1$ , and an integer *e*, returns a representation of the normalized floating point number equivalent to  $d \times 10^e$
- (mantissa *v*): given *v* as the representation of a normalized floating point number, returns its mantissa, expressed as a signed decimal number whose absolute value either is 0 or is  $\geq .1$  and  $< 1$ .
- (exponent *v*): given *v* as the representation of a normalized floating point number, returns its exponent, expressed as a signed integer

1. Using this abstract concept of floating point numbers, and assuming **nothing** about how floating point numbers are represented, implement the following procedures:

- (Add-Float *v1 v2*): Takes two non-negative normalized floating point numbers *v1* and *v2* as arguments and returns their sum as a normalized floating point number.
- (Multiply-Float *v1 v2*): Takes two non-negative normalized floating point numbers *v1* and *v2* as arguments and returns their product as a normalized floating point number.
- (Least-Upper-Bound *v*): Takes a normalized floating point number *v* as argument and returns the smallest integer that is greater than or equal to *v*.
- (Greater *v1 v2*): Takes two normalized floating point numbers *v1* and *v2* as arguments and returns true if the first is larger than the second, false otherwise.
- (Display-D *v*): Takes a normalized floating point number *v* as argument and displays it as a decimal number.
- (Display-E *v*): Takes a normalized floating point number *v* as argument and displays it as

$$s.ddddd * 10Eree$$

where *s* is the sign of the number, *dddd* are the five digits in the mantissa, *r* is the sign of the exponent, and *ee* is the absolute value of the exponent.

Store these procedures in a file called *float*.

2. As a representation for floating point numbers, use a dotted-pair representing the mantissa and exponent. Implement the constructors and selectors for this representation. Store these procedures in a file called *imp1*.
3. As a representation for floating point numbers, use an eight digit integer, where the rightmost two digits of the integer give the magnitude of the exponent of the floating point number, the third digit from the right is 0 if the exponent of the floating point number is positive and 1 if it is negative, the next five digits give the mantissa, and the sign of the eight-digit number gives the sign of the mantissa. Implement the constructors and selectors for this representation. Store these procedures in a file called *imp2*.

4. Show that your procedures stored in file *float* for operating on floating point numbers work with each of these underlying representations. **Remember that the procedures for operating on floating point numbers should not care what the underlying representation of the floating point numbers is!** Do this by loading files *float* and *imp1* and running test cases to show that you can add, multiply, display, etc. floating point numbers. Then load files *float* and *imp2* and run the same test cases to show that you can still add, multiply, display, etc. floating point numbers with this different underlying representation and **with no changes to the procedures in file *float* and with no changes to your test cases.**

**Hints:** It probably will be helpful to write the following procedures for use by your floating point procedures.

(Shift-Left *m i*) Takes as arguments a decimal number *m* and an integer *i* and returns *m* with its decimal point shifted *i* positions to the left.

(Shift-Right *m i*) Takes as arguments a decimal number *m* and an integer *i* and returns *m* with its decimal point shifted *i* positions to the right.

**The representation as a pair is easier to implement. You might try it first.**

**Extra Credit:**

There are two extra credit options. You may do one or both, if you wish. Extra credit will be awarded only if the procedures are entirely correct; there will be no partial extra credit.

1. **Divide-Float *v1 v2*:** Takes two non-negative normalized floating point numbers and returns the quotient of *v1* divided by *v2*, represented as a normalized floating point number.
2. All of the previous procedures working with negative as well as positive floating point numbers. (To get extra credit for this part, you must do all of the procedures, including **Divide-Float**, so that they work for both positive and negative floating point numbers.)

**Submission for Grading:** You should develop your own test cases — this is so that you gain more experience in testing procedures. Near the due date, the electronic submission system will become available for submitting Part-I of the project. To submit Part-I of the project, please do the following:

1. Copy your code from files *imp1* and *float* into the lab submission system for Project-1a and submit it.
2. Copy your code from files *imp2* and *float* into the lab submission system for Project-1b and submit it.
3. The only difference between your submissions for Project-1a and Project-1b should be your code for the procedures **make-normal-float**, **mantissa**, and **exponent**. Any other differences will result in a very low grade on Part-1 of the project.
4. If doing either of the extra credit options, submit your extra credit code into the lab submission system under Project-Extra-1a and Project-Extra-1b.