CS2035B Assignment 1: Testing Integer and Floating Point Conversions

Table of Contents

Identification

Robert Moir: 12345467890

Binary to Integer to Binary Conversions

```
format compact

s = ['11111111'; '10101010'; '10000000'; '00000000'];
for i=1:size(s,1)
    disp(['Testing: ' s(i,:)])
    val = int2bin8(bin2int8(s(i,:)))
    if (strcmp(s(i,:),val))
        disp('Pass')
    else
        disp('Fail')
    end
end
```

```
Testing: 11111111
val =
11111111
Pass
Testing: 10101010
val =
10101010
Pass
Testing: 10000000
val =
10000000
Pass
Testing: 00000000
val =
00000000
```

```
Pass
```

Integer to Binary to Integer Conversions

```
x = [28, -72, 128, -127];
for i=1:length(x)
    disp(['Testing: ' num2str(x(i))])
    val = bin2int8(int2bin8(x(i)))
    if (x(i) == val)
        disp('Pass')
    else
        disp('Fail')
    end
end
```

```
Testing: 28
val =
    28
Pass
Testing: -72
val =
   -72
Pass
Testing: 128
val =
   128
Pass
Testing: -127
val =
  -127
Pass
```

Decimal to Float to Decimal Conversions

In this case, I have implemented special tests to test "corner cases", i.e., cases that are not the main case the algorithm deals with. The special tests are followed by a generic test for generic input.

```
zero = '00000000000000000000000000000000';
posinf = '01111111100000000000000000000000';
neginf = '11111111100000000000000000000000';
s = ['01111111110000000000000000000000';
     '11111111110000000000000000000000';
     '00000000010101010101010101010101';
     '10000000110000000000000000000000'];
s(end+1,:) = dec2bin32(Inf);
s(end+1,:) = dec2bin32(10*pi);
s(end+1,:) = dec2bin32(10*pi+0.000002);
% Testing on number larger than Inf, should output Inf:
disp(['Testing: ' s(1,:)])
val = dec2bin32(bin2dec32(s(1,:)));
if (strcmp(val,posinf))
    disp('Pass')
```

```
else
    disp('Fail')
end
% Testing on number smaller than -Inf, should output -Inf:
disp(['Testing: ' s(2,:)])
val = dec2bin32(bin2dec32(s(2,:)));
if (strcmp(val,neginf))
    disp('Pass')
else
    disp('Fail')
end
% Testing a subnormal number, should output 0:
disp(['Testing: ' s(3,:)])
val = dec2bin32(bin2dec32(s(3,:)));
if (strcmp(val,zero))
    disp('Pass')
else
    disp('Fail')
end
% Generic test
for i=4:size(s,1)
    disp(['Testing: ' s(i,:)])
    val = dec2bin32(bin2dec32(s(i,:)))
    if (strcmp(s(i,:),val))
        disp('Pass')
    else
        disp('Fail')
    end
end
```

```
Testing: 01111111110000000000000000000000
Pass
Testing: 11111111110000000000000000000000
Pass
Testing: 00000000010101010101010101010101
Pass
Testing: 10000000110000000000000000000000
val =
10000000110000000000000000000000
Pass
Testing: 01111111100000000000000000000000
val =
01111111100000000000000000000000
Pass
Testing: 01000001111110110101001111010001
val =
01000001111110110101001111010001
Pass
Testing: 01000001111110110101001111010010
val =
01000001111110110101001111010010
Pass
```

Float to Decimal to Float Conversions

In this case I have allowed the corner cases to produce failed tests and then explained why the result we obtain is expected in a comment at the end.

```
format long
x = [2^128, 2^127, 2^-127, 2^-126, 10*pi, 31.415927, 31.415928];
% Convert input to single precision numbers
single(x)
for i=1:length(x)
    disp(['Testing: ' num2str(x(i))])
    val = single(bin2dec32(dec2bin32(x(i))))
    if (single(x(i)) == single(val))
        disp('Pass')
    else
        disp('Fail')
    end
end

% Comment on two failed tests:
% We expect 2^-127=5.8775e-39 to fail because this is a subnormal
 number,
% which is set to zero by our algorithm, so the output behaviour is
% correct
% We also expect 31.415927 to fail because the closest single
 precision
% number is larger, but our algorithm does not implement correct
 rounding,
% instead it rounds the number down to the next smallest single
 precision
% number. Hence the single precision number we return is smaller than
 the
% correct representation of 31.415927 in single precision.
% In contrast, 31.415928 is larger than the nearest single precision
% number, so we get this one right because our algorithm rounds down,
 which
% is correct in this case.

ans =
  1×7 single row vector
   1.0e+38 *
  Columns 1 through 6
         Inf   1.7014118   0.0000000   0.0000000   0.0000000
 0.0000000
  Column 7
   0.0000000
Testing: 3.402823669209385e+38
val =
  single
   Inf
Pass
Testing: 1.701411834604692e+38
val =
```

```
  single
   1.7014118e+38
Pass
Testing: 5.8775e-39
val =
  single
     0
Fail
Testing: 1.1755e-38
val =
  single
   1.1754944e-38
Pass
Testing: 31.4159
val =
  single
  31.4159260
Pass
Testing: 31.4159
val =
  single
  31.4159260
Fail
Testing: 31.4159
val =
  single
  31.4159279
Pass
```

Published with MATLAB® R2016b