Justin Bahr
NUID: 001592707

High Performance Computing
Homework 3 - Question 1

I ran my program on the COE Vector Linux platform (Intel ® Xeon ® CPU E5-2698 v4 @ 2.20GHz, 20 cores per socket, 2 sockets, with 791957684 KB of memory, and Linux 4.18.0).

a. Result shown below:

```
Single Precision Floating Point Calculations:
sin(0.05) = 0.049979917264699935913085937 5
sin(0.5) = 0.47942551970481872558593750 0
sin(1) = 0.84147095680236816406250000
sin(1.5) = 0.99732506275177001953125000

Double Precision Floating Point Calculations:
sin(0.05) = 0.04997916927067833775533145512781629804521799087524414062 5
sin(0.5) = 0.47942553860418707367685442477522883564233779907226562500 0
sin(1) = 0.84147096908663920888926668339991010725498199462890625000 0
sin(1.5) = 0.99732494917210323670531124662375077605247497558593750000 0

|Single Precision Error|:
sin(0.05): 3.376e-09
sin(0.5): 1.890e-08
sin(1): 2.801e-08
sin(1.5): 1.699e-04

|Double Precision Error|:
sin(0.05): 6.939e-18
sin(0.5): 1.593e-14
sin(1): 1.572e-08
sin(1.5): 1.700e-04
```

Figure 1. Taylor Series Sin(x) Approximations with Floats and Doubles

The implementation with doubles provides answers with more decimal places specified. This is because double precision floating point values have a longer mantissa. Additionally, the absolute error of results was consistently greater when single precision floats were used. Since not all decimal numbers can be exactly specified using IEEE 754 floating point, there are often rounding errors in their binary representations. These rounding errors are more severe for single precision because of the 23 bit mantissa compared to the 52 bit mantissa of a double precision floating point value. When many computations are made, these errors compound. Therefore, when accuracy is crucial and/or there is a multi step calculation, double precision floating points will produce a more desirable result in most cases.

b. IEEE 754 Floating Point Representations

- 2.1
    a. Single Precision
    Sign Bit: 0 (+)
    Exponent: 1 + 127_bias = 128 = 10000000_2
    Mantissa: 1.05 - 1_implied = 0.05 = 0. $0000110\overline{0110}_2$
    ≈ 0. 00001100110011001100
    Binary: $01000000000001100110011001100110_2$
    Hexadecimal: 40066666

    b. Double Precision
    Sign Bit: 0 (+)
    Exponent: 1 + 1023_bias = 1024 = 10000000000_2
    Mantissa: 1.05 - 1_implied = 0.05 = 0. $0000110\overline{0110}_2$
    ≈ 0. $0000110011001100110011001100110011001100110011001101_2$
    Binary:
    $0100000000000000110011001100110011001100110011001100110011001101_2$
    Hexadecimal: 4000CCCCCCCCCCCD

- 6300
    a. Single Precision
    Sign Bit: 0 (+)
    Exponent: 12 + 127_bias = 139 = 10001011_2
    Mantissa: 1.5380859375 - 1_implied = 0.5380859375
    = 0. 10001001110000000000_2
    Binary: $01000101110001001110000000000000_2$
    Hexadecimal: 45C4E000

    b. Double Precision
    Sign Bit: 0 (+)
    Exponent: 12 + 1023_bias = 1035 = 10000001011_2
    Mantissa: 1.5380859375 - 1_implied = 0.5380859375
    = 0. $10001001110000000000000000000000000000000000000000_2$
    Binary:
    $0100000010111000100111000000000000000000000000000000000000000000_2$
    Hexadecimal: 40B89C0000000000

- -1.044
    a. Single Precision
    Sign Bit: 1 (-)
    Exponent: 0 + 127_bias = 127 = 01111111_2
    Mantissa: 1.044 - 1_implied = 0.044 = 0. $0000101101000011100101 01\ldots_2$
    ≈ 0. 00001011010000111001011
    Binary: 10111111100001011010000111001011
    Hexadecimal: BF85A1CB

    b. Double Precision
    Sign Bit: 1 (-)
    Exponent: 0 + 1023_bias = 1023 = 01111111111_2
    Mantissa: 1.044 - 1_implied = 0.044
    = 0. $0000101101000011100101011000000100000110001001001110\ldots_2$
    Binary:
    10111111111100001011010000111001010110000001000001100010010011 10
    Hexadecimal: BFF0B4395810624E