Michael Timbes
Date: 08/30/2016
Class: Numerical Analysis

1 Largest Integer

The largest integer will be found by adding up as many numbers as possible until the number can't be represented any longer or the number stays the same.

```
INTEGER A,B
DO 1 A = 2,10
        B = B**A
        Print*,B
1 END DO
```

The max value will be limited to the processor and the type. In this case the type is INTEGER so the greatest value will be limited to the size of that type.

```
OUTPUT:
4
64
16777216
0
0
0...
```

If I change the type to a REAL, then the range of that value will be at the range of the largest that the processor can represent. Since I am running a 32 bit virtual machine the max value will be different than if the same program runs on a 64 bit processor.

```
4.000000000
64.00000000
16777216.0
1.32922800E+36
Infinity
Infinity....
```

In this case, the largest value that can be represented is $1.32922800 * 10^{36}$.Realistically there is no real representation for infinity. Most likely this is an addition to the compiler to tell the user that there is overflow that can't be represented any longer.

2 Smallest Real

The idea behind writing code to find the smallest real is to basically follow the process to find the largest real but inverted so that the value is one divided by the next number in the sequence.

```
INTEGER :: A
REAL :: B
DO 1 A = 2,10
        B = 1/(B**A)
        Print*,B
1 END DO
```

The smallest number has the same restrictions as other numbers. That being a combination of the processor and the data type. In this case it seems that the smallest number that the REAL type can represent is $9.31322575 * 10^{-10}$.

```
csci2000@csci2000-VirtualBox:~$ ./a.out
  0.250000000
   6.25000000E-02
   1.56250000E-02
   3.90625000E-03
   9.76562500E-04
   2.44140625E-04
   6.10351562E-05
   1.52587891E-05
   3.81469727E-06
   9.53674316E-07
   2.38418579E-07
   5.96046448E-08
   1.49011612E-08
   3.72529030E-09
   9.31322575E-10
      Infinity
```

3 Sum Going Forward

The basic idea of this program is to have three variables: 'A' to store the current sum, 'OLD' to store the previous sum, and 'B' as the loop count and the 'i' in the equation given in problem three. The loop continues until the 'A' and the 'OLD' sum are the same value.

```
PROGRAM ADDSUM

REAL :: A = 0,OLD = 0 !A is the running sum, OLD is the older sum
INTEGER :: B

DO B = 1,10000000

OLD = A
A= A + (1/float(B))
```

```
      IF (A - OLD == 0) THEN
PRINT*, A
STOP
END IF

END DO
PRINT*,"DONE"
END PROGRAM ADDSUM
```

The output of the program is the following. The sums before are omitted since it took $1.0 * 10^7$ iterations to produce this output.

```
 csci2000@csci2000-VirtualBox:~$ gfortran a1p3.f95
csci2000@csci2000-VirtualBox:~$ ./a.out
 CONDITION A - OLD == 0 MET. RESULT:
  2097152
   15.4036827
   15.4036827
```

4 Sum Going Reverse

Using the same number of values but going in reverse produced a different number than the same going in forward. It is known that this output is more accurate because the smallest terms are added first to not have as much affect in the rounding of the sums.

```
csci2000@csci2000-VirtualBox:~$ gfortran a1p3.f95
csci2000@csci2000-VirtualBox:~$ ./a.out
 CONDITION A - OLD == 0 MET. RESULT:
  2097152
   15.4036827
   15.4036827
csci2000@csci2000-VirtualBox:~$ ./a.out
   15.1328993
 DONE
```