# HW 5

## Luke Meiler

## Problem 1

We are given the series $f_n = f_{n-1}^2$ to implement in C++. However, before implementing it we are asked to select three C++ data types and determine the maximum n that the selected data type can store.

We will choose the int, long int, and float data types. In order to determine the maximum n that can be achieved, we look to the maximum value that these types can attain. These are as follows: 32,767, 65,535, and 2,147,483,647 for int, unsigned int, and long int, respectively. This means that we simple need to determine how many times we need to square the number 2 before it surpases each of these numbers.

Using a regular scientific calculator, one can see that an int data type can store up to $n = 3$. The zeroth element is given as 2, squaring this gives the first element as 4, the second as 16, the third as 256, and the fourth would be 65,536, but this is outside of the scope of an int. For our second data type, the unsigned int, the overflow cap is 65,535. This would be able to store up to $n = 3$ by this same logic. For our third data type, the long int, the overflow cap is 2,147,483,647, meaning it could store up to $n = 4$.

## Problem 2

For the second problem, we need to actually implement our solution to problem one in C++. Secondly, we need to write a method that will return the value of the variable at each step of the series.

I personally found it easier to write the function in python first, and then convert it to C++. Found below are three functions: the python implementation of the first part of the problem, the C++ implementation of the first part of the problem, and the C++ implementation for the second part of the problem.

Each of these functions takes an integer for the maximum value that the data type we are interested in can take. These were listed earlier in the notebook. This means that we can use these functions for any data type as long as we know what the maximum value that the data type can take is.

For the first part of the problem, the function will return an integer representing the maximum $n$ value. For the second part of the problem, the function will return the value of the series at a given step.

Each one of these functions is repeated using the long int data types as well.

```
!cat ./Series.c++
```

```
// def findN(overflow):
//     n = 0
//     fN = 2
//     while(fN**2 < overflow):
//         fN = fN**2
//         n += 1
//         print(fN,n)
//     return n

#include <cmath>
#include <list>

extern "C" int findN(int overflow) {
    int n = 0;
    int fN = 2;
    while (pow(fN,2) < overflow) {
        fN = pow(fN, 2);
        n = n + 1;
    }
    return n;
}

extern "C" int findNStep(int overflow, int step) {
    int n = 0;
    int fN = 2;
    int count = 0;
    while (count < step) {
        count = count +  1;
        fN = pow(fN, 2);
    }
    if (fN > overflow) {
        return 0;
    }
    else {
        return fN;
    }
}

extern "C" long int findNLong(long int overflow) {
    long int n = 0;
    long int fN = 2;
    while (pow(fN,2) < overflow) {
        fN = pow(fN, 2);
        n = n + 1;
    }
    return n;
}

extern "C" long int findNStepLong(long int overflow, long int step) {
    long int n = 0;
    long int fN = 2;
    long int count = 0;
    while (count < step) {
        count = count +  1;
        fN = pow(fN, 2);
    }
    if (fN > overflow) {
        return 0;
    }
    else {
        return fN;
    }
}
```

## Problem 3

### Generating the Library

For the third problem, we use ctypes to generate a shared library of problem 2, import this library, and run the code in this notebook to see if it works.

Beginning with generating the library, we use some code that was mentioned in class to generate the Series library. Immediately following this, the second command is run to generate the ".so" shared library. Finally, we display the three files necessary for this library that we've made: the C++ file, the library, and the shared library.

```
!gcc  -fPIC -c ./Series.c++ -o ./Series.o -std=c++11
```

```
!gcc -shared ./Series.o -o ./Series.so -lstdc++
```

```
!ls -lah ./Series.*
```

```
-rw-r--r-- 1 compphys compphys 1.2K Nov  1 02:51 ./Series.c++
-rw-r--r-- 1 compphys compphys 2.6K Nov  1 02:51 ./Series.o
-rwxr-xr-x 1 compphys compphys  16K Nov  1 02:51 ./Series.so
```

### Importing and Running the Library

Now that the shared library has been made, we can now import it and ctypes to actually run the C++ code that was written before. First, we import ctypes and the shared library that was just made.

```
from ctypes import *
```

```
series = cdll.LoadLibrary("./Series.so")
```

Now we can run the code through a number of tests to ensure that it works correctly. We selected the int, long in, and float data types before, so we can check these now. Int has a maximum value of 32,767, long int has a maximum value of 65,535, and long int has a maximum value of 2,147,483,647.

```
series.findN(32767)
```

```
3
```

```
series.findN(65535)
```

```
3
```

```
series.findN(214748347)
```

```
4
```

As we can see, these return the $n$ value that we want for these three cases. Next, we can see if the implementation to return the specific value of the series at a given step is functional. For instance, this first example is taking an unsigned int and asking for the second value in the series.

```
series.findNStep(65535, 2)
```

```
16
```

We correctly get 16, $((2^2)^2)$. Previously, we saw that unsigned ints can only go 3 steps in the series, so if we input a 4 we should get the overflow value of 0 encoded into the function.

```
series.findNStep(65535, 4)
```

```
0
```

We see here that the correct value was returned. For the sake of thoroughness, all three data types will be shown to correctly function in this way. This works as intended since the C++ code was compiled into python and now works exactly as it would in C++.

```
series.findNStep(32767, 1)
```

```
4
```

```
series.findNStep(32767, 4)
```

```
0
```

```
series.findNStep(214748347, 4)
```

```
65536
```

```
series.findNStepLong(214748347, 5)
```

```
0
```