Homework 5

Austin Marga

Problem 1:

Suppose you need to compute the series:

$$f_n = f_{n-1}^2$$

Given that $f_0 = 2$, what is the maximum n that can be stored in three C++ data types?

Let's pick the following:

- unsigned int -> Min is 0 -- Max is 65,535
- int -> Min is -32,768 -- Max is 32,768
- long int -> Min is -2,147,483,648 -- Max is 2,147,483,647

Let's list the first few values of n

- $f_0 = 2$
- $f_1 = 4$
- $f_2 = 16$
- $f_3 = 256$
- $f_4 = 65,536$
- $f_5 = 4,294,967,296$

From this, we can see that:

- For unsigned int: $n = 3$
- For int: $n = 3$
- For long int: $n = 4$

Problem 2

First, let's attempt to write the first method in Python for clarity.

```
def findn(num):
    """
    This method takes some input number and returns the maximum n value that can be stored.
    num is intended to be the maximum size of a C++ data type.
    Input Argument: num must be a number.
    Returns: n is a number.

    """
    # The initial conditions are n = 0 and $f0  = 2$
    n = 0
    f0 = 2
    # We want to continue to compute the next step in the series
    # until the next step exceeds the input.
    while (f0**2 < num):
        f0 = f0**2
        n += 1
    return n
```

Let's test our cases for unsigned int, int, and long int

```
# Using the three data types as in Problem 1.
unsigned_int_n = findn(65535)
int_n = findn(32768)
long_int_n = findn(2147483647)
```

```
print("Unsigned_int_n should be 3: and it is" + " " + str(unsigned_int_n))
print("int_n should be 3: and it is" + " " + str(int_n))
print("long_int_n should be 4: and it is" + " " + str(long_int_n))
```

```
Unsigned_int_n should be 3: and it is 3
int_n should be 3: and it is 3
long_int_n should be 4: and it is 4
```

In `HW5.c++`, we have the methods that cover both parts of Problem 2.

```
!cat ./HW5.c++
```

```
#include <cmath>
#include<iostream>
using namespace std;
/*
This line "using namespace std;" is needed for the print statement. Found from
https://stackoverflow.com/questions/15185801/cout-was-not-declared-in-this-scope
*/

extern "C" int findn(int num){

    /*
    This method takes some input number and returns the maximum n value that can be stored.
    num is intended to be the maximum size of a C++ data type.

    Input Argument: num must be a number.
    Returns: n is a number.
    */
    int n = 0;
    int f0 = 2;
    while (pow(f0,2) < num) {
        f0 = pow(f0,2);
        n = n + 1;
    }
    return n;
}
int unsigned_int_n = findn(65535);
int int_n = findn(32768);
int long_int_n = findn(2147483647);

int Print() {
    /*
    This is how you could do cases in C++, I will do them in Python instead to demonstrate
    how to use methods from a (shared) library as in Problem 3.
    */

    cout << "For the cases of unsigned int, int, and long int" <<endl;
    cout << "For unsigned int, n should be 3: n = " <<  unsigned_int_n << endl;
    cout << "For int, n should be 3: n = " <<  int_n << endl;
    cout << "For long int, n should be 4: n = " <<  long_int_n << endl;
    return 0;
}

extern "C" int value_at_n(int num, int n){
```

```
    /*
    This method takes some input number "num" and the step in the series
    and returns the value of the series for at step n.

    num is intended to be the maximum size of a C++ data type.
    If the value at the given n value is larger than num (the size of the type),
    it will return 0.

    Input Argument: num must be a number. n must be a number.
    Returns: value is a number.
    */
    int f0 = 2;
    int value = pow(f0,pow(2,n));
    if (value > num){
        return 0;
    }
    else{
        return value;
    }
}
```

Problem 3

Here, I use `ctypes` to generate a shared library from the C++ file that I used for Problem 2 - `HW5.c++` The specific lines of code to do this was found from the Notebook found in the CompPhys folder `ctypes_functions.ipynb`.

First, we have to generate a `.o` file, which is the machine code from the compiling phase.

```
!gcc  -fPIC -c ./HW5.c++ -o ./HW5.o -std=c++11
```

Next, we turn it into a shared `.so` file.

```
!gcc -shared ./HW5.o -o ./HW5.so -lstdc++
```

Using the `ctypes` library, we can load our libary (our shared file) to use the methods inside.

```
from ctypes import *
```

```
func = cdll.LoadLibrary("./HW5.so")
```

Now, let's make sure the test cases we have using the C++ method work correctly as in the case for Python.

```
unsigned_int_n_c = func.findn(65535)
int_n_c = func.findn(32767)
long_int_n_c = func.findn(2147483647)

print("Unsigned_int_n should be 3: and it is" + " " + str(unsigned_int_n_c))
print("int_n should be 3: and it is" + " " + str(int_n_c))
print("long_int_n should be 4: and it is" + " " + str(long_int_n_c))
```

```
Unsigned_int_n should be 3: and it is 3
int_n should be 3: and it is 3
long_int_n should be 4: and it is 4
```

Hooray! It worked!

Now, to test the second method, `value_at_n` This method returns the value of the recursive function at a specific step. We should test it for when n is within the size of the type and when it is not.

```
Step_3 = func.value_at_n(65535,3)
print("For n=3 given an unsigned int, we expect to get the value of 256, and we get " + str(Step_3))
```

```
For n=3 given an unsigned int, we expect to get the value of 256, and we get 256
```

```
Step_4 = func.value_at_n(65535,4)
print("n = 4 is out of bounds for this type.")
print("For n = 4 given an unsigned int, we expect to get the value of 0, and we get " + str(Step_4))
```

```
n = 4 is out of bounds for this type.
For n = 4 given an unsigned int, we expect to get the value of 0, and we get 0
```

Double Hooray! They both worked!