Homework 5

Problem 1

Suppose you need to compute the series:

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

If the value $f_0 = 2$, what is the maximum   that can be stored in C++ data types? (9p) Choose three different data types as an example and use the data types sizes used in the docker image for your hardware platform. (3p) Document how you got to the solution. (9p)

Tip for writing equations in Markdown. In Markdown, if we want to write the subscript n-1, put a curly bracket around n-1, so like this f_{n-1}. Similarly, if we want to write 2^16 in Markdown, we surroung 16 with the curly brackets too! Here we can see the difference:

2^{16} with curley brackets around 16

2^16 without curley brackets around 16

Back to the problem. I will choose these three data types: Unsigned int, float and double. They are common data types in C++, so let's investigate them. Using the example from datatypeexamples.ipynb, we can probably use the ctype function sizeof to investigate the size of each of these three data types.

```
from ctypes import *

#Why the asterisk sign at the end of the command? I do not know,
#If the asterish sign is not there, the command will return with error.
```

```
sizeof(c_uint())
```

```
4
```

```
sizeof(c_float())
```

```
4
```

```
sizeof(c_double())
```

```
8
```

The sizof() function returns the number of bytes in the data type. Each byte contains 8 bits typically, and each bit can be 0 or 1. if let's say a datatype have just 1 byte assigned to it, then the maximum number it can store is 11111111, we have eight 1's, and that correponds to the number 255 in decimal system. Or we can express the number 255 as $2^8 - 1$. So 2 raised to the power of the number of bits in the data type minus 1 should tell us the mamimum number that can be recorded by that data type. :)

For our problem, $f_0 = 2$. So let's see some examples and get an idea of the pattern:

$f_1 = f_0^2 = 2^2$

$f_2 = f_1^2 = (2^2)^2 = 2^4$

$f_3 = f_2^2 = ((2^4)^2) = 2^8$

$f_4 = f_3^2 = ((2^8)^2) = 2^{16}$

$f_5 = f_4^2 = ((2^{16})^2) = 2^{32}$

So I think the pattern is $f_n = 2^{2^n}$

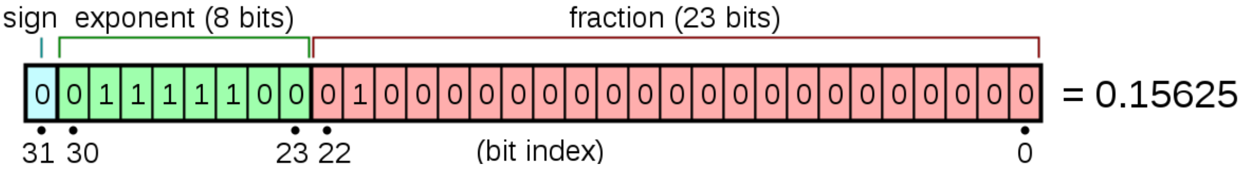
For unsigned integer, it has 4 bytes in it, so the maxinum number it can record is $2^{4\times8}-1=2^{32}-1$. It couldn almost record f_5, but unfortunately it's smaller by just 1! So the maximum n that be stored on unsigned intger is 4, in other words, f_4 is the biggest number in the series it can record.

Float and double data types are different from unsigned integer! They are called Floating-Point Types. Our earlier logic about 11111111 being the largest number that can be held with 1 byte doesn't work here.

Essentially, a floating point data can be thought of as a number expressed in scientific notation, for example

1.2345×10^2

The bits in the memory will be assigned different roles, one of them will record the sign of the number (positive, 0, or negative, 1); a number of them will record the number itself, although I don't exactly know how to record all the decimal places!? And a number of them will record the exponent. Maybe some bits will also record the base of the exponential! It is complicated. Here is an image from wikipedia entry on floating point arithmatic https://en.wikipedia.org/wiki/Floating-point_arithmetic, and it illustrates what we just said.



What is certain out of these complexities is this, the floating point data is not always an integer, that must be very clear.

I feel that it's a complex task to estimate the range for the floating point data types, so I'll just refer to what others say about their ranges. :P According to this website: https://www.tutorialspoint.com/cprogramming/c_data_types.htm, the range for c_float is 1.2E-38 to 3.4E+38, and the range for c_double is 2.3E-308 to 1.7E+308. These are some really big numbers. I kind of regret picking these two data types now.

By trial and error, $2^{128}=3.4028237e+38$ and $128=2^7$

Oh, so the range for c_float really isn't that big! It can record up to only f_7, in fact I think f_7 is bigger than its upper limit, so it can only record f_6. The maximum n c_float can store is only 6.

(There must be a fixed number of bits assigned to represent the exponent in each data type, I guess the the number of bits assigned to represent the exponent in c_float is 7...)

The upper limit for c_double, 1.7E+308, is roughly equal to 2^{1024} and $1024=2^{10}$. So I think f_{10} is just out of reach for c_double. It can record f_9, in other words, the maximum n c_double can store is 9.

(Again, I'm guessing that the number of bits assigned to represent the exponent in c_double is 10.)

The answers:

c_uint, $n=4$

c_float, $n=6$

c_double, $n=9$

Problem 2

Fork the homework repository (1p):

https://github.com/ubsuny/CP1_PythonCpp

Write a C++ method to demonstrate Problem 1 on your computer (12p). That is, implement the series with variables using your three data types (in case you didn't do P1 choose three C++ data types now). Write (an)other method(s) that returns the value of the variable (with the appropriate data type) at each step of the series (8p).

Hint: You can start from one of the C++ examples in "ReviewCpp".

I'm not really good at programing, so I had to look up some example codes online, and modify those codes :P The series we want is a recursive series, so I looked up examples of c++ loops, and I came across a code that generates the Fibonacci series: https://www.programiz.com/cpp-programming/examples/fibonacci-series As we know the fibonacci series is recursive, so that's exactly what I need :D

In fact the code to generate the Fibonacci series is more complicated, since in the Fibonacci series, each term is dependent on the values of TWO preceding terms, whereas in our series, each term is only dependent on the value of the previous term. So with some changes, here is my code:

```cpp
#include <iostream>
using namespace std;

int main() {
    int n;
    double t1 = 2, nextTerm = 0;  #This program works with the c++ double data type

    cout << "Enter the largest value of n: ";
    cin >> n;

    cout << "Square Series: ";

    for (int i = 0; i <= n; ++i) {
        // Prints the first term.
        if(i == 0) {
            cout << t1 << ", ";
            continue;
        }

        nextTerm = t1 * t1;

        t1 = nextTerm;

        cout << nextTerm << ", ";
    }
    return 0;
}
```

We can't run this code in Jupyter notebook yet, since it only runs python codes. In order to run a c++ code we need to compile it, which is shown in the next problem of this homework, problem 3.

We can understand what is happening in this code. We first define an integer n, which is the subscript for our term f_n, in fact it is the subscript of the last f_n we want to display, obviously this is an infinite loop, so we need to inform the code where to stop :)

The double t1 is our first term in the series, so it is the value of f_0, and as we know, $f_0 = 2$. Also we can see in this code, we use the data type c_double :)

The code asks us for the largest n we want:

```
cout << "Enter the largest value of n: ";
cin >> n;
```

Then it is a standard for loop in c++

for (initialization; condition; update)

{ // body of-loop }

The integer i keeps track of the terms we are generating. It goes from 0, and increase with steps of 1. So i is the subscript of each term f_i. We define the first term in the series first, $f_0 = t1$, then we update the value of each term with the value of the previous term.

To change the data type, we just modify this line:

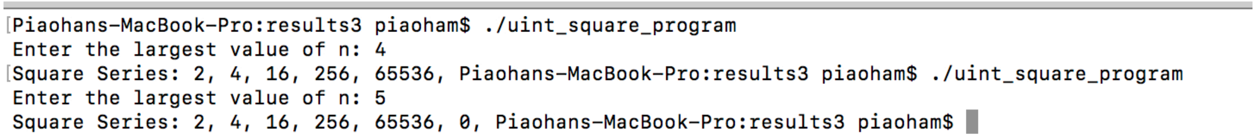
double t1 = 2, nextTerm = 0;

Replace double with any other c++ data type we want. For instance, here are the codes for unsigned integer and float:

```
#include <iostream>
using namespace std;

int main() {
    int n;
    unsigned int t1 = 2, nextTerm = 0; #This program works with the c++ unsigned integer data type

    cout << "Enter the largest value of n: ";
    cin >> n;

    cout << "Square Series: ";

    for (int i = 0; i <= n; ++i) {
        // Prints the first term.
        if(i == 0) {
            cout << t1 << ", ";
            continue;
        }

        nextTerm = t1 * t1;

        t1 = nextTerm;

        cout << nextTerm << ", ";
    }
    return 0;
}
```

```
#include <iostream>
using namespace std;

int main() {
    int n;
    float t1 = 2, nextTerm = 0;  #This program works with the c++ float data type

    cout << "Enter the largest value of n: ";
    cin >> n;
    cout << "Square Series: ";

    for (int i = 0; i <= n; ++i) {
        // Prints the first term.
        if(i == 0) {
            cout << t1 << ", ";
            continue;
        }

        nextTerm = t1 * t1;

        t1 = nextTerm;

        cout << nextTerm << ", ";
    }
    return 0;
}
```

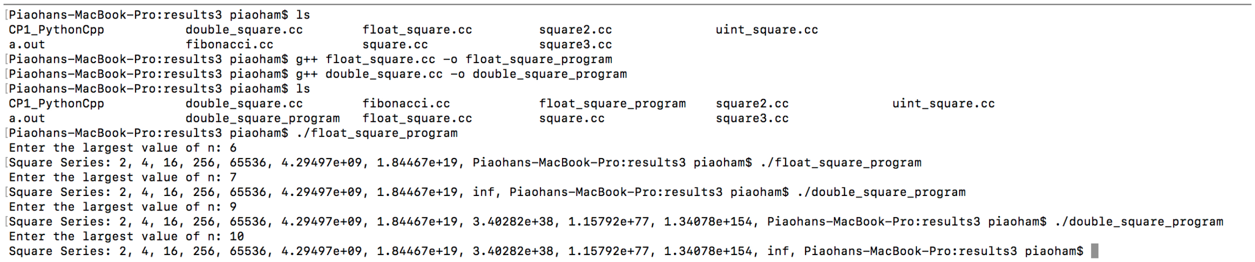
These c++ codes are all tested on my own computer, and they work! :D Here are some screenshots from my computer.



```
[Piaohans-MacBook-Pro:results3 piaoham$ ./uint_square_program
 Enter the largest value of n: 4
[Square Series: 2, 4, 16, 256, 65536, Piaohans-MacBook-Pro:results3 piaoham$ ./uint_square_program
 Enter the largest value of n: 5
 Square Series: 2, 4, 16, 256, 65536, 0, Piaohans-MacBook-Pro:results3 piaoham$ 
```

Our answer in problem 1 seems correct! The maximum n that can be stored on c_uint is 4, and if I ask the program to run to f_5, then it returns the f_5 value as 0, it glitches out as we expected :)

Here are the results for the c_float and c_double data types. I also included the steps of how we compile a c++ code using the compiler g++.



```
[Piaohans-MacBook-Pro:results3 piaoham$ ls
 CP1_PythonCpp           double_square.cc        float_square.cc         square2.cc              uint_square.cc
 a.out                   fibonacci.cc            square.cc               square3.cc
[Piaohans-MacBook-Pro:results3 piaoham$ g++ float_square.cc -o float_square_program
[Piaohans-MacBook-Pro:results3 piaoham$ g++ double_square.cc -o double_square_program
[Piaohans-MacBook-Pro:results3 piaoham$ ls
 CP1_PythonCpp           double_square.cc        fibonacci.cc            float_square_program    square2.cc              uint_square.cc
 a.out                   double_square_program   float_square.cc         square.cc               square3.cc
[Piaohans-MacBook-Pro:results3 piaoham$ ./float_square_program
 Enter the largest value of n: 6
[Square Series: 2, 4, 16, 256, 65536, 4.29497e+09, 1.84467e+19, Piaohans-MacBook-Pro:results3 piaoham$ ./float_square_program
 Enter the largest value of n: 7
[Square Series: 2, 4, 16, 256, 65536, 4.29497e+09, 1.84467e+19, inf, Piaohans-MacBook-Pro:results3 piaoham$ ./double_square_program
 Enter the largest value of n: 9
[Square Series: 2, 4, 16, 256, 65536, 4.29497e+09, 1.84467e+19, 3.40282e+38, 1.15792e+77, 1.34078e+154, Piaohans-MacBook-Pro:results3 piaoham$ ./double_square_program
 Enter the largest value of n: 10
 Square Series: 2, 4, 16, 256, 65536, 4.29497e+09, 1.84467e+19, 3.40282e+38, 1.15792e+77, 1.34078e+154, inf, Piaohans-MacBook-Pro:results3 piaoham$ 
```

The maximum n that can be stored on c_float is 6, and if I ask the program to run to f_7, then it returns the f_7 value as infinity! The program is pretty smart, it knows the value exceeds its range, so it just calls it inifinity :)

The maximum n that can be stored on c_float is 9, and if I ask the program to run to $f_1$0, then it returns the $f_1$0 value as infinity.

Problem 3 (21 points)

Use either ctypes or SWIG to generate a (shared) library of problem 2 (7p) (In case you didn't do problem 2 choose any C++ example from the "ReviewCpp" folder). Add a documentation

how you did that (7p). Import that library into a Jupyter notebook and use it to document if you see what you expect (and / or compare with problem 1). Why or why not? (7p)

I tried to use ctype, since that is the method I went through during the class.

I used the ctype_setuptools.ipynb as my template.

First we write a setup python file. I named it setup2.py, and it looks like this:

```
from setuptools import setup, Extension

# Compile *double _square2.cpp* into a shared library
setup(
    # the square series code has to end in cpp for it to work.
    ext_modules=[Extension('square', ['double_square2.cpp'],
                          language="C++",),],
)
```

We have to modify our original c++ code a little, and make sure it ends with .cpp, instead of .cc or anything else. Only .cpp file works somehow.

The modified c++ code looks like this, and it is renamed double_square2.cpp

```
/*
The keyword [ extern "C" ] is used to declare functions in C++ to be compiled in C. The compiler will u
*/


    #include <cstdio>

extern "C"  int Square() {
    int n = 4;
    double t1 = 2, nextTerm = 0;


    for (int i = 0; i <= n; ++i) {
        // Prints the first term.
        if(i == 0) {
            printf("  %f\n", t1);
            continue;
        }

        nextTerm = t1 * t1;

        t1 = nextTerm;

        printf("  %f\n", nextTerm);
    }
    return 0;
}
```

As you can see I replaced all the std:cout() entries with printf() functions, the ctype importer doesn't like std:cout(). Also I need to add the line #include , in order to use print() function

Most importantly! I have to put extern ”C” right in front of my main function int Square(). Then the ctype importer can identify the function int Square(). Otherwise it would say ’Square is not defined’.

Next we run setup2.py with the following command. The exclamation mark means this command is run in the terminal.

```
!python setup2.py build
```

```
running build
running build_ext
building 'square' extension
creating build
creating build/temp.linux-x86_64-3.9
x86_64-linux-gnu-gcc -pthread -Wno-unused-result -Wsign-compare -DNDEBUG -g -fwrapv -O2 -Wall -g -ffile-
creating build/lib.linux-x86_64-3.9
x86_64-linux-gnu-gcc -pthread -shared -Wl,-O1 -Wl,-Bsymbolic-functions -Wl,-Bsymbolic-functions -Wl,-z,
```

Yah! A bunch of stuff, this means the c++ code has been successfully compiled! Otherwise it would tell you where the errors are.

Then we find the output file, in other words, the executable file, using this command

```
!ls -lah build/lib.linux-x86_64-3.9
```

```
total 24K
drwxr-xr-x 3 compphys compphys  96 Oct 31 19:55 .
drwxr-xr-x 4 compphys compphys 128 Oct 31 19:55 ..
-rwxr-xr-x 1 compphys compphys 21K Oct 31 19:55 square.cpython-39-x86_64-linux-gnu.so
```

So, out executable file is named ”square.cpython-39-x86_64-linux-gnu.so”. Next we import it using ctypes.

```
import ctypes
import numpy as np

csquare = ctypes.cdll.LoadLibrary("./build/lib.linux-x86_64-3.9/square.cpython-39-x86_64-linux-gnu.so")
```

Nothing happens when we press shift + enter, so we are good. It is almost imported now.

Next we defined the input for our code.

```
csquare.Square.restype = ctypes.c_int
csquare.Square.restype = ctypes.c_double

csquare.Square.argtypes = [ctypes.c_int, ctypes.c_double, ctypes.c_double]
```

Nothing happens again when we press shift + enter, now we know the code is now fully imported!! If we had not put extern ”C” right in front of the main function int Square() in the cpp code, it would return with error in this step, ”Square is not defined”, thankfully we corrected that mistake, and it works fine.

Time to reap the reward, let’s run the code!

```
result = csquare.Square(4, 2, 0)

print(result)
```

```
65536.0
```

We specified our input (4, 2, 0), and this means (f_4 is what we want, $f_0 = 2$ initially, some more initialization)

And indeed if $f_0 = 2$, then $f_4 = 65536$. And we defined our variable to be c_double, so it has a decimal place, as we would expect! Now we're happy.