HW5

Problem 1

We have to compute the sum of the series : $f_n =f_{n-1}^2 $. Lets compute the first few value for this function: Given $ f_1 =f_0^2 = 2^2 = 4 $ therefore, $f_2 =f_1^2 = 4^2 = 16 $ $f_3= f_2^2 = 16^2 = 256 $ ..... We can see that for any given number of 'n' the sum is going to be : S = $ f_0^2 + f_1^1 + ...+f_{n-1}^2 $

Different data typye has different limits. The information about data types are taken from Wikipedia/C-data_types . The examples and codes (which i have modified for my problem) are taken from G4G.org.

Choosing maximum 'n' for our data types

For my hardware (**ARM64**) the maximum number can be stored in c++ data types are given below: For "int" : 2147483647 For "float":$ 3.4028*210^{38} $ and for "double": $1.7* 10^{308} $. Previously I put wrong name for the data types.

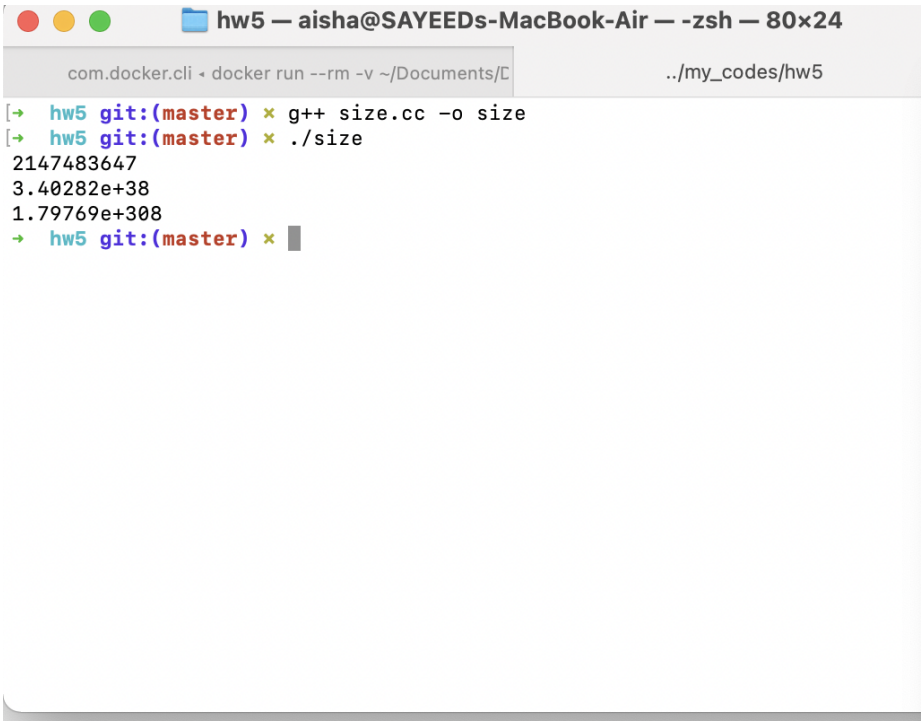
I ran (executed) the following code to find out the max supported number for the c++ data types :

#include

```
int main() {
using namespace std;
cout << numeric_limits<int>::max() << endl;
cout << numeric_limits<float>::max() << endl;
cout << numeric_limits<double>::max() << endl;

return 0;
```

}

Which showed me the following output in my computer : 2147483647 3.40282e+38



1.79769e+308

We can see that for our series the sum for any chosen n must not exceed these limits. We can find out the maximum allowed number for our data types as follows : For "int" : $ S_{max} $ = 2147483647 , thus n (max) would be = 5 ; otherwise the last value of the sum will exceed the max limit For "float":$ S_{max} $ = 3.40282e+38 , thus n (max) would be n = 7 ; otherwise the last value of the sum will exceed the max limit For "Double" : $ S_{max} $ = 1.79769e+308, thus n (max) would be n = 10 ; otherwise the last value of the sum will exceed the max limit.

Problem 2

Solve problem 1 with C++

We will wrtite 3 different codes for 3 different data types (int, float and double) . For the **"int""** types the codes are given follows:

#include

using namespace std;

int main()

{ int n,init_s=2, sum = 0;

```
cout << "Please enter the value for n (max 7) : ";
```

```
cin >> n;


for (int i=1; i<n; i++)
```

{

```
init_s=init_s*init_s;
```

sum = sum+init_s; }

cout « "The sum of the series for n = " « n « " is : " « sum ;

return sum;

}

we save this file as **series_int.cc** in current folder and run the following command to make it excecutable : **g++ series_int.cc -o series_int** than we run the file using **./series_int** . In my computer the code runs without any issue and its shows the expected result.

image

For the Float data type I will use the following code :

#include

using namespace std;

#include

typedef std::numeric_limits< double > dbl; // Declaring maximum precision

int main()

{ float n,init_s=2, sum = 0;

```
cout << "Please enter the value for n  : ";


cin >> n;

for (int i=1; i<n; i++)


{
```

init_s=init_s*init_s;

sum = sum+init_s;

}

cout.precision(dbl::max_digits10); // Making sure to print max precision.

cout « "The sum of the series for n = " « n « " is : " « sum ;

return sum; }

image

we save this file as **series_float.cc** in current folder and run the following command to make it excecutable : **g++ series_int.cc -o series_float** than we run the file using **./series_float** . In my computer the code runs without any issue and its shows the expected result.

For the double types we wrtie the following code:

#include

using namespace std;

#include

typedef std::numeric_ limits< double > dbl;

int main()

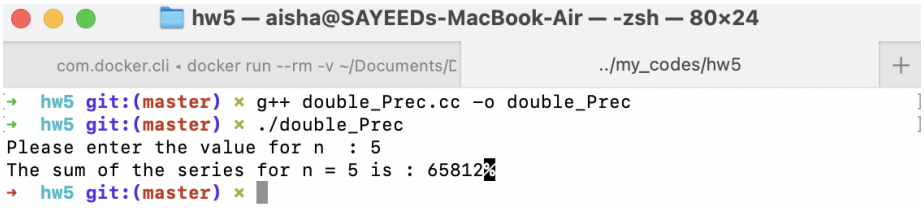
{ double n,init_s=2, sum = 0;

```
cout << "Please enter the value for n  : ";

cin >> n;


for (int i=1; i<n; i++)

                    {

init_s=init_s*init_s;

sum = sum+init_s;

                    }


 cout.precision(dbl::max_digits10); // Making sure to print max precision.

 cout << "The sum of the series for n = " << n << " is : " << sum ;
```

return sum; }

we save this file as **double_prec.cc** in current folder and run the following command to make it excecutable : **g++ double_prec.cc -o double_prec**

than we run the file using **./double_prec** . The result shown in my terminal is given below :



Therefore , I think my code gave the correct result .

Its not a big change from my previus attempt , but I think this version will not loose much of precision.