## A3Q1 - Iteration Calculation

Calculate the value of **e** by following iteration:

$$\sum_{n=0}^{\infty} \frac{1}{n!} = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \frac{1}{5!} + \cdots$$

Input a double-precision float number which represents a precision threshold.

Your program should terminate the iteration when the difference between two successive iterations is smaller than the precision threshold.

Print the value of e (as double-precision float).

NOTES: all the numbers in the calculation are taken as double.

### Sample Testcases

Input #1	Сору	Output #1	Сору
10		1.0	
Input#2	Сору	Output #2	Сору
1		2.0	
Input#3	Сору	Output #3	Сору
0.01		2.70833333333333	
Input #3	Сору	Output #3	Сору
0.01		2.70833333333333	
Input #4	Сору	Output #4	Сору
0.0000248015873015873		2.71827876984127	

## A3Q2 - IEEE745 Format

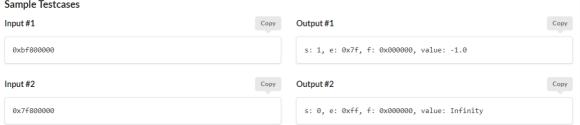
Get the input string (whose length is not exceed 10), take it as a single-precision floating number's IEEE745 code in hexadecimal, output its sign bit, exponent (in hexadecimal), fraction part (in hexadecimal) of IEEE745 code and its value.

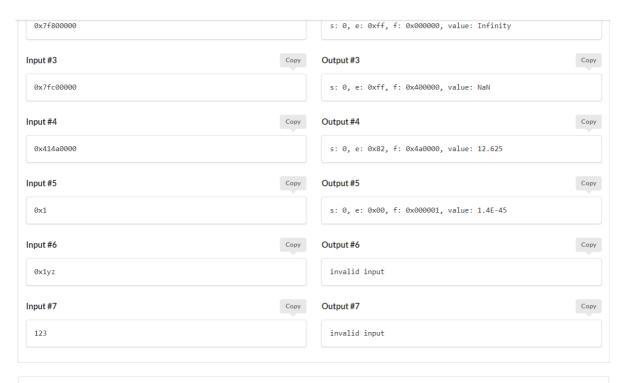
If the input string is invalid, print the error info "invalid input" and exit the program.

## NOTES:

- If the length is smaller than 10, take the string as the lower part of code of single float number, complete the higher part of the code with 0 s.
- If the first two character of the input string is neither <code>@x</code> nor <code>@X</code>, output " invalid input ".
- If there is invalid hexadecimal character in the input string, output "invalid input".
- It is suggested to use "print float "syscall to print the value.

# Sample Testcases





## A3Q3 - Division

Define a macro which is named as "divSCheck" with three parameter (%f1, %f2, %f3), it is expected to implement following function:

- 1. Firstly check the value in register %f3 , if it is **zero**, set the value of register \$k0 to be 11 , trig the exception.
- 2. Secondly divider the single-precision floating data in register %f2 by the single-precision floating data in register %f3 and store the result to the register %f1.

#### NOTES:

The code of Q3 and Q4 would be used in the following code:

```
1 .text
2 li $v0,6
3 syscall
4 mov.s $f20,$f0
5 li $v0,6
6 syscall
7 mov.s $f21,$f0
8 divSCheck($f12,$f20,$f21)
9 li $v0,2
10 syscall
11 li $v0,10
12 syscall
```

Before or after the use of "divSCheck", there would be some registers in Coprocessor1 be used. (Tips: using Stack to protect the 'old' value in this registers before change its)



# A3Q4 - Exception Handler

Define an exception handler which is sensitive with the exception event (the exception code is 11 in register \$k0 ), in this situation it report the exception reason "exception: divisor is 0.0" then return to the normal procession.

### NOTES:

The code of Q3 and Q4 would be used in the following code:

```
1 .text
2 li $v0,6
3 syscall
4 mov.s $f20,$f0
5 li $v0,6
6 syscall
7 mov.s $f21,$f0
8 divSCheck($f12,$f20,$f21)
9 li $v0,2
10 syscall
11 li $v0,10
12 syscall
```

Before or after the use of "divSCheck", there would be some registers in Coprocessor1 be used. (Tips: using Stack to protect the 'old' value in this registers before change its)

#### Sample Testcases Input #1 Сору Output #1 Сору 1.0 2.0 0.5 Input #2 Сору Output #2 Сору 1.0 0 exception:divisor is 0.0 Infinity Input #3 Сору Output #3 Сору exception:divisor is 0.0 NaN 0 0