CS 383: Programming Project 2
Due: December 9, 11:59 pm

Collaboration Policy. Assignments will be done individually: each student must hand in their own answers. It is acceptable for students to collaborate in understanding the material but not in solving the problems or programming. Use of the Internet is allowed, but should not include searching for existing solutions.

Under absolutely no circumstances code can be exchanged between students. If some code was shown in class, it can be used, but it must be obtained from Canvas, the instructor or the TA.

Assignments from previous offerings of the course must not be re-used. Violations will be penalized appropriately.

Late Policy. No late submissions will be allowed without consent from the instructor. If urgent or unusual circumstances prohibit you from submitting a homework assignment in time, please e-mail me.

Deliverable. Submit on Canvas a **zip** file containing:

1. a **pdf** file describing your approach,
2. a file with the source code named taylor.s (not executables, project files etc.).

Requirements

1. Implement the following Taylor series approximation to the exponential function:

$$e^x = \sum_{n=0}^{\infty} \frac{1}{n!} x^n = 1 + \frac{1}{1!}x + \frac{1}{2!}x^2 + \frac{1}{3!}x^3 + ...$$

2. Your program must accept two inputs in the `.data` segment of the program: a **double** precision floating point number representing x and an integer representing the number of terms in the approximation.
3. Your program must include a function that computes the value of the i^{th} term of the series. Make sure that function returns properly.
4. Your program must print out the final result. See the example below.

Hints

1. Doubles must be in the D registers (see below) and cannot be mixed with integers. You can see the double registers under *SIMD, Double* in Eclipse.

2. You can use `MUL` for integer multiplication, `FMUL` for double multiplication and `FDIV` for double division.

3. `SCVTF` (Signed fixed-point Convert to Floating-point) can be used to convert integers to doubles.

4. You can use a loop or a recursive function to compute the factorial.

5. You can use `"The approximation is %f\n"` or something along these lines for printing out the result. The first `%f` argument must be in D0, etc.

The example below may be useful.

```
 .text
 .global main
 .extern printf
main:
 SUB sp, sp, #16
 STR x30, [sp]

 // multiply theta with itself
 LDR x2, =theta
 LDR d2, [x2]
 FMUL d0, d2, d2
 LDR x0, =prt_str1
 // calling printf alters registers X0-X7
 // save them in stack to use again
 SUB sp, sp, #16
 STR d0, [sp]
 STR d2, [sp, #8]
 BL printf
 LDR d0, [sp]
 LDR d2, [sp, #8]
 ADD sp, sp, #16

// convert 5 to 5.0
// and multiply with theta
LDR x2, =n
LDR x1, [x2]
SCVTF d1, x1
FMUL d0, d1, d2
LDR x0, =prt_str2
BL printf

// return to caller
LDUR x30, [sp]
```

```
ADD sp, sp, #16
BR x30

 .data
n:
.quad 5
theta:
.double 0.45
prt_str1:
.ascii "theta square is %f \n\0"
prt_str2:
.ascii "five times theta is %f \n\0"
 .end
```