First Recursive Function: Factorial

- **Example: Factorial**
  - The factorial function can be written explicitly as follows:

```
int fac(int n)
{
   if (n == 0)
      return(1);                  <--- easy case
   else
   {
      return( m * fac(n-1) );     <--- Return the solution for fac(n)
   }
}
```

  - The factorial function is called with a statement that is similar to the following:

```
int n, result;

result = fac(n);
```

- **The stack frame structure for the factorial function**
  - The **stack frame structure** that you need to created **depends** on the **number** of parameter variables and the number of local variables used in the function.
  - In the above example:
    - the factial function has **1 parameter variable** and
    - **0 local variables**.
  - The stack frame structure created will looks as follows:

```
+----------------------+ <------------ Frame Pointer (A6) and Stack pointer (A7)
| Saved Frame Pointer |
+----------------------+
|  Return Address      |
+----------------------+
| use for parameter n |
+----------------------+
|     .......          |
|  rest of the stack   |
|     .......          |
```

We **do not** need to be concerned with the **"rest of the stack"** (i.e., the part of the stack **used by other functions**) because **this function** has **no business** with any of the information stored in that area of the stack !

(In fact, if you **do** mess with the data stored in the "rest of the stack area", you will have an extraordinary painful experience with programming recursive function in assembler as you try to debug your recursive function.

So recursion resembles a cat - curiosity will kill it...)

- **The factorial program**
  - **Factorial in assembler:**

```
* ===============================================
* main: result = fac(4)
* ===============================================
Start:  move.l n, -(a7)
        bsr    fac           ; fac(4)

        adda.l  #4,a7        ; pop useless parameter from stack
        move.l d0, result    ; Put 4! in result

Stop:   nop

n:      dc.l 4
result: ds.l 1

---------------------------------------------------------
*
* ===============================================
* int fac(int n)
* {
*    if (n == 0)
*       return(1);
*    else
*    {
*       return (n * fac(n-1));
*    }
* }
*    Input: n on stack
*    Output: n! in register d0
* ===============================================
*
fac:
********************************* PRELUDE
        move.l a6, -(a7)     ; Save caller's frame pointer
        move.l a7, a6        ; Setup my own frame pointer
        suba.l #0, a7        ; No local variables (you can omit this instruction)
*********************************
* ------------------------------ ; Testing n == 0....
        move.l 8(a6), d0     ;
        cmp.l  #0, d0        ; n == 0 ??
        bne    Else          ;

* ------------------------------ ; Then....
        move.l #1, d0        ; then part: return 1 in D0

********************************* POSTLUDE
        move.l a6, a7         ; Deallocate local variables
        move.l (a7)+, a6     ; restore caller's frame pointer
*********************************
        rts

* ------------------------------ ; Else....
Else:
        move.l 8(a6), d0     ;
        sub.l #1, d0         ; D0 = n - 1
*
* ------------------------------ ; fac(n) is calling fac(n-1) now !!!!
*
        move.l d0, -(a7)     ; Push (n-1) as parameter
        bsr     fac            ; Call fac(n-1)
        adda.l #4,a7           ; Clean up parameter from stack

*                              ; Note: d0 contains the result of fac(n-1) !!!

        move.l 8(a6), d1       ; d1 = n
        muls   d1, d0          ; d0 = n*fac(n-1)

*                              ; Now we are ready to exit... Watch the stack !

********************************* POSTLUDE
        move.l a6, a7          ; Deallocate local variables
        move.l (a7)+, a6       ; restore caller's frame pointer
*********************************
        rts
```

- ○ The complete example can be found in the following assembler program file: click here
- ○ You may want to get the following Debug file for EGTAPI to use with it: click here
- ○ I will highlight certain steps in the program in the remainder of the webpage....

- **Passing parameter n from main program to fac**

  The main program passes the parameter n to factorial by pushing n onto the system stack with the following instruction:

```
    move.l n, -(a7)
```

  This will create the following stack structure:

```
   +----------------------+ <------------ Stack pointer (A7)
   |     parameter n      |
   +----------------------+
   |     .......          |
   |  rest of the stack   |
   |     .......          |
```

- **Main program calling fac function**

  The main program calls the factorial function with a **bsr** instruction:

```
    bsr  fac
```

  This will create the following stack structure:

```
   +----------------------+ <------------ Stack pointer (A7)
   |     return address   |
   +----------------------+
   |     parameter n      |
   +----------------------+
   |     .......          |
   |  rest of the stack   |
   |     .......          |
```

- **Prelude of the Factorial function:**

  The prelude of the factorial function consists of the 3 instructions:

```
********************************* PRELUDE
        move.l a6, -(a7)     ; Save caller's frame pointer
        move.l a7, a6        ; Setup my own frame pointer
        suba.l #0, a7        ; No local variables
*********************************
```

  I will explain what each one does below. Make sure that you realise that the structure of the stack frame is like this when the prelude is **always** executed:

```
+---------------------+ <------------ Stack pointer (A7)
|     return address  |
+---------------------+
|     parameter n     |
+---------------------+
|     .......         |
|  rest of the stack  |
|     .......         |
```

- move.l a6, -(a7)

  This will save the frame pointer on the stack, creating this partial stack frame structure:

  ```
  +---------------------+ <------------ Stack pointer (A7)
  |     saved a6        |
  +---------------------+
  |     return address  |
  +---------------------+
  |     parameter n     |
  +---------------------+
  |     .......         |
  |  rest of the stack  |
  |     .......         |
  ```

- move.l a7, a6

  This will make the frame pointer A6 points to the stack frame that is now being built:

  ```
  +---------------------+ <---- Frame pointer A6 & Stack pointer (A7)
  |     saved a6        |       point to the same location....
  +---------------------+
  |     return address  |
  +---------------------+
  |     parameter n     |
  +---------------------+
  |     .......         |
  |  rest of the stack  |
  |     .......         |
  ```

- suba.l #0, a7

  This instruction does nothing to the stack pointer A7... (we could omit it)

  ```
  +---------------------+ <---- Frame pointer A6 & Stack pointer (A7)
  |     saved a6        |       point to the same location....
  +---------------------+
  |     return address  |
  +---------------------+
  |     parameter n     |
  +---------------------+
  |     .......         |
  |  rest of the stack  |
  |     .......         |
  ```

- When the prelude is finish, the stack frame is complete and the actual function can begin.

- **How to access the parameter in fac:**

  - Parameter n is located 8 bytes **below** starting from the address contained in the frame pointer A6.

    So the address mode that will let you get to this variable is 8(A6)

- **How factorial calls itself:**

  It is no different from how the main program calls the factorial function. Simply push the parameter on the stack, and call factorial.

But **make sure** you **pop the parameter n** from the stack after factorial returns - because the parameter has not been cleaned up.

The following is the program fragment where factorial calls fac(n-1):

```
       move.l 8(a6), d0    ; retrieve parameter n into register d0
       sub.l #1, d0        ; d0 = n - 1
*
* ----------------------------- ; fac is calling fac now !!!!
*
        move.l d0, -(a7)    ; Push (n-1) as parameter
        bsr    fac          ; Call fac(n-1)
        adda.l #4,a7        ; Clean up parameter from stack

*                           ; NOTE: return value of fac(n-1) in register D0
```