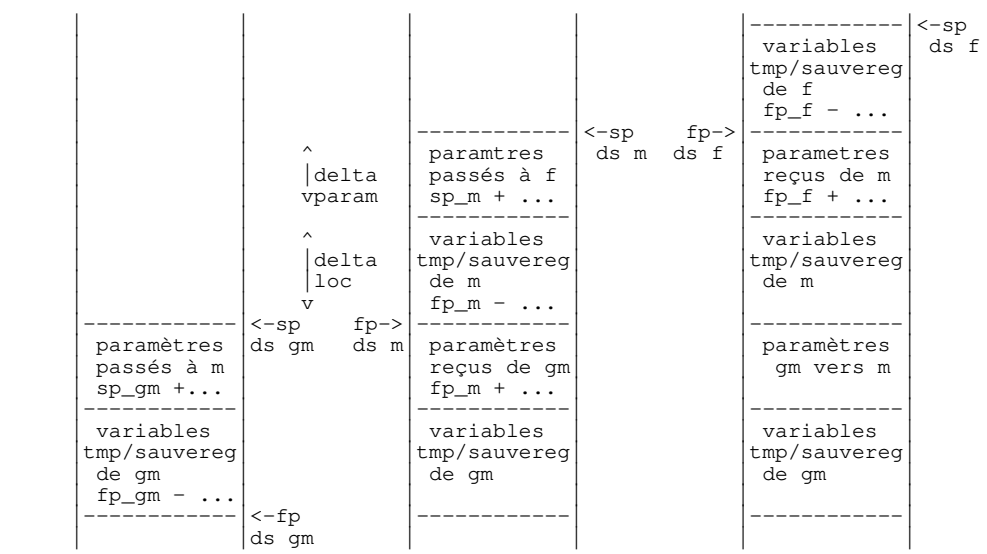
```
Procédures (cas général)

I) Gestion des résultats

1.1) Retourner un seul résultat : fonction

On ajoute un mot au bloc de paramètres passés.

void f (void)   int  g(int x)
{               {
int y;          return x+3;
y = g(3);       }
}

x_de_g =3; -------->  resultat_de_g = x_de_g+3;
y = resultat_de_g <----

1.2) Paramètres valeur/résultat ou plusieurs résultats

Comment faire en sorte qu'on puisse passer en paramètre une variable à
modifier ?

On utilise un/des paramètre(s) de type pointeur.
Avec l'adresse contenue dans le pointeur on peut lire ou modifier la
variable passée.

short int y;
short int z;   // z pourrait aussi être une variable locale de f

// convention d'appel : paramètres dans regs : p = r0, m = r1,
// ip/r12 = tmp non sauvegardé par l'appelée.

void f(void)   void pm (short int *p, short int *m)
{              {
               *p = *p + 1;
pm(&z,&y);     *m = *m - 1;
y --;           }
}              // y décrémentée deux fois au final.

p_de_pm = &z_de_f;      (ldr r0,= z)
m_de_pm = &y;           (ldr r1,= y)
           ----------> *p_de_pm = *p_de_pm + 1;
                       (ldrsh r10,[r0], add r10,r10,#1; strh r10,[r0])
                        *m_de_pm = *m_de_pm - 1;
                       (ldrsh r10,[r1], sub r10,r10,#1; strh r10,[r1])
y--;        <----------

Et si la variable à modifier est un pointeur ?
--> utiliser un paramètre de type pointeur de pointeur ...

int x;
int *ptr;

void f(void)          void g (int **p)
{                     {
g(&ptr);              *p = &x;
}                     }

II) Principe de gestion de la récursion

Exemple :  0! = 1; n! = n * (n-1)!
```



```
Récursion directe : une fonction s'appelle elle-même à nouveau.
        indirecte : f -> g -> ...  -> f

unsigned long facto (unsigned long n)
{
unsigned long f = n;
if (n <= 1)
  f = 1;
else
  f = f * facto(n-1);
return f;
}

main --->
         facto(4)
         f=4       -->
                     facto (3) -->
                     f=3
                                   facto(2)
                                   f=2
                                            --> facto(1)
                                            <-- f = 1
                                <-- f = 2
                    <-- f = 6
     <-- f = 24

Une allocation DYNAMIQUE d'un bloc de mémoire (paramètres reçus + locaux)
par APPEL. Propriété LIFO : dernier alloué premier libéré.

--> on stocke des blocs dans un grand tableau appelé la pile.
    allocation lors de l'appel et libération lors du retour

III) Allouer et empiler, libérer et dépiler

registre sp (stack pointeur/sommet de pile) repère la limite alloué/libre.

Variantes de schéma de pile : fd, fa, ed, ea
  full descending, full ascending, empty desceding, empty ascending
  --> pile croit vers adresses hautes/basses
  --> sommet de pile pointe dernière case pleine/première case vide

ARM : fd  (très répandue)

Allouer(t) : sp = sp -t       empiler(r)  : sp = sp -4; Mem[sp] = r
Libérer(t) : sp = sp +t       r=depiler() : r = Mem[sp]; sp = sp + 4

stmfd sp!,{r0,r3-r5}  : empiler(r5);empiler(r4);empiler(r3);empiler(r0)
ldmfd sp!,{r0,r3-r5}  : depiler(r0);depiler(r3);depiler(r3);depiler(r5)
           ------->     toutjours en ordre croissant
```

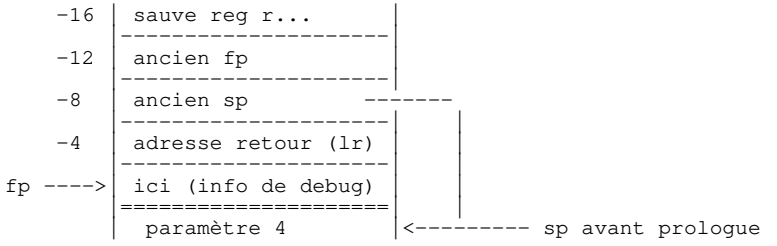
```
III) Stockage des locaux et des paramètres dans une pile

Registre sommet de pile (sp) : repère limite plein/vide
Registre pointeur de cadre (fp) : repère bloc précédent (appelante)

   ^                   appel de m                appel de f
   | moins             --------->                --------->
   |                     allouer                   allouer
   | plus
   V                 retour de m               retour de f
                     <----------               <----------
                        libérer                   libérer
```




```
Remarque : réutilisation de l'espace mémoire.


a appelle b puis a appelle c : zone allouée par b réallouée à c.

IV) Squelette de codage

fonction:  @ prologue
           @ sauvegarder fp  (en sp - ....)
           @ fp = sp
           @ sauvegarder autres registres (en fp - ....)
           @ sp = sp - delta_loc
           @ ceci réalise l'allocation de mémoire des locaux

           @ corps
           @ acces aux locaux en fp - ..., paramètres reçus en fp + ...
                 @ appel de g
                 @ allocation des paramètres : sp = sp - delta_param
           @ écrire paramètres de g en sp + ...
           @ bl g
                 @ libérer : sp = sp + delta_param

           @ épilogue
```



```
           @ restaurer tous les registres
           @ --> rétablit au passage anciens fp et sp --> libération
           @ mv pc,lr

Remarque : on pourrait faire sans fp, mais la position relative à sp des locaux
           et paramètres reçus change à chaque allocation de bloc de paramètres.

Variante   : on alloue au début une seule fois le plus grand bloc de paramètres
             nécessaire parmi tous les appels de fonction dans le corps,
             et on libère une seule fois à la fin.

V) Optimisations

Pour économiser des accès à la pile, convention répandue :
* n premiers arguments d'appels dans les registres (ARM : 4 dans r0 à r3)
* résultat d'une fonction stocké à la place du premier argument
* adresse de retour :
   + RISC : dans un registre (ARM :lr)/sauvée par appelée
   + CISC : empilée par instruction d'appel dans l'appelante (jsr de 68000)

VI) Code standard de gnu/ARM

  Particularité : on fait fp <- sp - 4
  Les registres sont sauvés par l'appelée, excepté ip.

  Code standard du prologue :

prologue:  mov   ip,sp
           stmfd sp!,{fp,ip,lr,pc}
           sub   fp,ip,#4
ici:       sub   sp, sp, #DELTA
           str   rx, [fp, #-Delta_sauve_rx]
           ...
           str   rz, [fp, #-Delta_sauve_rz]
```




```
  Code standard de l'épilogue
           ldr   rz, [fp, #-Delta_sauve_rz]
           ...
           ldr   rx, [fp, #-Delta_sauve_rx]
           ldmea fp, {fp, sp,pc}
           @ restaure anciens fp et sp, restaure lr dans pc (fait mov pc,lr)

  A noter : lors d'un appel, penser à sauver r0 à r3 qui contenaient les
  paramètres reçus et contiendront les paramètres passés.
```