```
Mécanisme d'appel de procédure (cas simple sans récursion)

I) Branchements aller et retour : un exemple

Procédures : même traitement à plusieurs endroits dans programme
             un seul exemplaire du code (corps de la procédure) + branchements

int y = 2;
int t;

void gm (void)    void m (int x)     void f (int a, int b, int c)
{                 {                  {
int z;            int vloc;          int y;     /* différent de y global */
z = 3;            int v;             y = a + b;
f (3,4,5);        f(x+2,y,3);        t = y + c;
m (2);            vloc++;            }
z++;              }
f (1,0,2*z);
z = z + 4;
}

           gm appelante de m et de f
           m appelante de f et appelée de gm
           f appelée de gm et m

Séquence
     z = 3;
     a_de_f = 3;
     b_de_f = 4;
     c_de_f = 5;
         --------1-----------------------------
     x_de_m = 2;  <-------------------------- | ---------------------------------
         --3------->  a_de_f = x_de_m + 2;    |                                 |
                      b_de_f = y;             |                                 |
                      c_de_f = 3;             v
                             ------4---------> y_de_f = a_de_f + c_de_f;    |
                                               ^  t       = y_de_f + c_de_f;    |
                      vloc ++     <--------    | --5---     ----------------2--
     z++   <-----6------                       |          |
     a_de_f = 1;                               |          |
     b_de_f = 0;                               |          8
     c_de_f = 2*z;   -7------------------------           |
     z = z+4;       <-----------------------------------

A la fin d'une procédure, le branchement de retour ne va pas toujours à la
même étiquette : l'adresse de retour (prochaine instruction à exécuter au
retour de l'appel) est un paramètre de l'appel.

Propriété LIFO : dernier appel - premier retour.

II) Contraintes et espaces de stockages

  N'importe quelle procédure est susceptible d'appeler n'importe quelle
autre dans n'importe quel programme (cf procédures de bibliothèque genre
printf).

  Une convention entre appelante et appelée doit préciser comment passer les
paramètres d'appel explicites (pour f ->  a,b,c : écrits par le programmeur) et
implicite (adresse de retour dans l'appelante : m ou gm).

  Les procédures sont écrites indépendamment les unes des autres. La seule
```



```
chose que deux procédures connaissent l'une de l'autre est la convention
d'appel.

  Pour chaque procédure (exemple ici m), quatre zones de stockage
(--> traitées comme des structures) :

  * variables globales partagées (data/bss)

  * paramètres (x et adresse de retour) reçus de l'appelante (gm)
      --> stockage partagé entre la procédure et ses appelantes

  * variables et temporaires locaux (privés) de la procédure (vloc)
    éventuellement de même nom que var globales (cas de y de f) ou locales
    d'autres procédures
       --> stockage privé, non partagé

  * paramètres passés (a,b,c) à une autre procédure (f) -> stockage partagé
avec la procédure appelée.

  Stockage à répartir entre registres et mémoire.

  Principe (sans récursion) : allouer statiquement à chaque procédure
    * Une structure regroupant les paramètres d'appels, que l'appelante
      remplira. Simplification : un mot complet/élément, même si 8 ou 16 bits.
      Optimisation : passer les n (n=4 pour ARM) premiers paramètres dans
des registres, les autres dans la structure.
      gcc ARM : repérée par registre sp/r13.

    * une structure privée regroupant les variables, temporaires et
    sauvegarde de registres
      gcc ARM : repérée par fp/r11

  Conflits d'utilisation des registres
    a) registre utilisé à la fois par appelante et par appelée.
    b) registre utilisé à la fois pour paramètre reçu et paramètre passé

          --> sauvegarde avant modification/restauration après utilisation
"l'appelante retrouve les registres du processeur dans le même état après appel"

Exemple : gm décide de stocker z dans r5
          f  décide de stocker y dans r5
      sans sauvegarde/restauration : y = ... dans f change z dans gm.

Technique usuelle : procédure appelée sauvegarde (dans prologue)/restaure
(dans épilogue) contenu des registres qu'elle modifie.
--> variante +rare : appelante sauve avant/restaure après appel registres
dont le contenu ne doit pas être perdu.


III) Instructions de branchement aller et retour

     Convention ARM : adresse de retour stockée dans registre r14/lr
                      "link register"

     Branchement retour : pc = adresse de retour contenue dans lr
                  mov  pc,lr

     Banchement aller :

     mov   lr,pc              ---           @ equivalent
     b     procedure            |           bl     procedure
suite:     add     r0, r0, r0 <--           add     r0, r0, r0
```

```
     Pc étant en avance de 2 instr, lr pointe sur l'instruction qui
suit le branchement (ici le add).


Pb si oubli de sauvegarde de lr en case d'appels en cascade.
--> bl f dans m écrase adresse de retour de m vers gm.
--> à la fin de m, mov pc,lr revient dans m après appel de f

IV) Propriété du passage de paramètre par valeur

  Les arguments de l'appels (ou paramètres réels) sont des expressions.
Chaque expression est évaluée par l'appelante et (une copie de sa valeur)
est déposée dans l'emplacement de stockage du paramètre formel.

arguments
valeur(paramètre réel)   déposée dans      paramètres formel
copie de valeur (x+2)      --->            a
copie de valeur (y)        --->            b
copie de valeur (3)        --->            c

  Si l'expression est une variable de l'appelante -> une copie de sa
valeur est affectée au paramètre formel.

  Une procédure peut-elle modifier la valeur de son paramètre formel ?
oui, si elle n'a plus besoin de la valeur passée par l'appelante.

  Si on ajoute  b = b+1 dans le corps de f, y est-il modifié ?
NON --> seule la COPIE de la valeur de y stockée dans b est modifiée.
La valeur de y est inchangée.
```