Travaux dirigés 4 : fonctions et procédures (1)

Correction. *Note aux chargés de TD : voir cours fonction. L'appel de fonction leur est présenté comme une expression qui s'évalue comme la valeur retournée par cette fonction. Pour cette semaine les appels de fonctions utilisateurs n'ont lieu que dans le* `main`*, à l'exception d'une des dernières questions du TD sur le coefficient binomial. On dit spécifiquement procédures pour les fonctions de type de sortie void.*

La trace des programmes avec appel de fonction doit représenter la pile d'appel (voir tableau **??***). Pour cela, au moment de l'appel d'une fonction, on fait la trace de cette fonction en indentant vers la droite. À la fin de l'exécution de la fonction, quand elle renvoie sa valeur, on poursuit la trace de la fonction appelante. La ligne responsable de l'appel est indiquée avant la trace de la fonction, mais son exécution proprement dite (si elle a sa place dans la trace) n'est réalisée qu'après l'appel, donc elle se retrouve après le retour de fonction. Notez que :*

1. *on ne trace que les fonctions utilisateurs (bien que l'on rappelle que les fonctions systèmes sont bien appelées, comme printf).*
2. *lorsque plusieurs fonctions utilisateurs sont appelées sur une même ligne (plusieurs fonctions, arguments d'une autre fonction), l'ordre d'évaluation ne doit pas jouer. On choisit les appels de gauche à droite, du plus imbriqué au moins imbriqué.*

---

1 Trace de fonctions

Faire la trace du programme suivant et dire ce que calcule la fonction `est_xxx`.

```
1  #include <stdlib.h> /* EXIT_SUCCESS */
2  #include <stdio.h> /* printf() */
3  /* Declaration constantes et types utilisateurs */
4  #define TRUE 1
5  #define FALSE 0
6
7  /* Declaration de fonctions utilisateurs */
8  int est_xxx(int n);
9
10 /* Fonction principale */
11 int main()
12 {
13     /* Declaration et initialisation des variables */
14     int n = 9;
15
16     if (est_xxx(n))
17     {
18         printf("L'entier %d est xxx\n", n);
19     }
20     else
21     {
```

```
22          printf("L'entier %d n'est pas xxx\n", n);
23      }
24
25      /* Valeur fonction */
26      return EXIT_SUCCESS;
27  }
28
29  /* Definition de fonctions utilisateurs */
30  int est_xxx(int n)
31  {
32      int i;
33
34      for (i = 2; i < n; i = i + 1)
35      {
36          if (n % i == 0)
37          {
38              return FALSE;
39          }
40      }
41      return TRUE;
42  }
```

Correction. *Cette fonction teste si son argument est un nombre premier : elle renvoie TRUE si son argument n est premier (ou négatif ou nul – remarque : zéro n'est pas premier), et FALSE sinon.*

`main()`

ligne	n	Affichage (sortie écran)
initialisation	9	
16		
22		L'entier 9 n'est pas xxx
26	renvoie EXIT_SUCCESS	

`est_xxx(9)`

ligne	n	i	Affichage
initialisation	9	?	
34		2	
40		3	
38	renvoie FALSE		

Correction.

2 Déclaration et définition de fonctions

Les fonctions `valeur_absolue()`, `factorielle()` et `minimum()` ne sont pas fournies avec le programme suivant. Compléter le programme avec le prototype et le code des fonctions (le `main` doit rester inchangé) et faire la trace du programme complet.

```
14  int main()
15  {
16      int x = -3;
17      int y = 5;
18      int z;
19
20      /* Un calcul sans signification particulière */
21      x = valeur_absolue(x); /* valeur absolue de x */
22      z = minimum(x, y);     /* minimum entre x et y */
```

```
23      z = factorielle(z);    /* z! */
24      z = minimum(y, z);     /* minimum entre y et z */
25
26      /* Valeur fonction */
27      return EXIT_SUCCESS;
28  }
```

Correction.

Code.

```
1   #include <stdlib.h> /* EXIT_SUCCESS */
2
3   /* Declaration de constantes et types utilisateurs */
4
5   /* Declaration de fonctions utilisateurs */
6   /* Retourne la valeur absolue de son argument entier */
7   int valeur_absolue(int n);
8   /* Retourne le minimum des deux valeurs en argument */
9   int minimum(int a, int b);
10  /* Retourne la factorielle de l'argument entier (si positif) */
11  int factorielle(int n);
12
13  /* Fonction principale */
14  int main()
15  {
16      int x = -3;
17      int y = 5;
18      int z;
19
20      /* Un calcul sans signification particulière */
21      x = valeur_absolue(x); /* valeur absolue de x */
22      z = minimum(x, y);     /* minimum entre x et y */
23      z = factorielle(z);    /* z! */
24      z = minimum(y, z);     /* minimum entre y et z */
25
26      /* Valeur fonction */
27      return EXIT_SUCCESS;
28  }
29  /* Definition de fonctions utilisateurs */
30  int valeur_absolue(int n)
31  {
32      if (n < 0) /* si n est négatif */
33      {
34          /* Valeur fonction */
35          return -n; /* inverser le signe de n et renvoyer */
36      }
37      else /* n est positif */
38      {
39          /* Valeur fonction */
40          return n;
41      }
42  }
43
44  int minimum(int a, int b)
45  {
46      if (a < b) /* Si a est le minimum */
47      {
48          /* Valeur fonction */
49          return a;
50      }
51      else /* a n'est pas le minimum */
52      {
```

```
53          /* Valeur fonction */
54          return b;
55      }
56  }
57
58  int factorielle(int n)
59  {
60      int i; /* Var. de boucle */
61      int res = 1; /* resultat */
62
63      for (i = 1; i <= n; i = i + 1) /* Pour i = 1, 2, ..., n */
64      {
65          res = res * i; /* mettre i dans le produit */
66      }
67
68      /* Valeur fonction */
69      return res;
70  }
71
```

Correction.

3 Écriture de fonctions

Pour les questions suivantes il faut donner la déclaration et la définition de chaque fonction. Vous pouvez faire l'exercice une première fois en donnant uniquement les déclarations, puis le reprendre pour les définitions. Répondre dans un seul programme, dans lequel vous écrirez une fonction principale (`main`) faisant appel à ces fonctions pour les tester.

1. Écrire la fonction `carre` qui prend en entrée un entier et qui renvoie le carré de cet entier.
2. Écrire la fonction `cube` qui prend en entrée un entier et qui renvoie le cube de cet entier.
3. Écrire la fonction `est_majeur` qui prend en entrée un entier représentant l'age en années d'une personne et renvoie `TRUE` si cette personne est majeure et `FALSE` sinon (on considérera les deux constantes utilisateurs bien déclarées).
4. Écrire la fonction `somme` qui prend en entrée un entier n et qui renvoie $\sum_{i=1}^{i=n} i$. Où faut-il déclarer la variable de boucle ?
5. Si vous ne l'avez pas fait à l'exercice précédent, écrire la fonction `factorielle` qui prend en entrée un entier n et qui renvoie $\Pi_{i=1}^{i=n} i$.
6. Écrire la procédure `afficher_rectangle` qui prend en entrée deux entiers, largeur et hauteur, et affiche un rectangle d'étoiles de ces dimensions.
7. Écrire la fonction `saisie_utilisateur` sans argument, qui demande à l'utilisateur de saisir un nombre entier et le retourne.
8. Écrire la fonction `binomial` qui prend en entrée un entier n et un entier p et retourne le nombre de tirages différents, non ordonnés et sans remise, de p éléments parmi n, c'est à dire le coefficient binomial :

$$\binom{n}{p} = \mathrm{C}_n^p = \frac{n!}{p!(n-p)!}.$$

   Faire appel à la fonction `factorielle`.

9. Optionnel. Écrire la fonction `saisie_dans_intervalle` à deux paramètre entiers a et b, qui demande à l'utilisateur de saisir un nombre entier jusqu'à ce qu'il soit dans l'intervalle $[a, b]$ et le retourne. On pourra fixer un nombre maximum de tentatives après quoi le nombre de l'intervalle le plus proche de la saisie utilisateur sera renvoyé.

Correction.

```
int cube(int x);
int est_majeur(int age);
int somme(int n);
void afficher_rectangle(int largeur, int hauteur);
int saisie_utilisateur();
int binomial(int n, int p);
int autre_somme(int n);


int carre(int x)
{
   return x*x;
}

int cube(int x)
{
   return x*x*x;
}

int est_majeur(int age)
{
    if (age < 18)
    {
        return FALSE;
    }
    /* sinon */
    return TRUE;
}

int somme(int n)
{
    int i;
    int somme = 0;

    for (i = 1; i <= n; i = i + 1)
    {
        somme = somme + i;
    }

    return somme; /* ou bien juste : return n * (n+1) / 2; */
}

void afficher_rectangle(int largeur, int hauteur)
{
    int i; /* lignes */
```

```
    int j; /* colonnes */

    for (i = 0; i < hauteur; i = i + 1) /* pour chaque ligne */
    {
        /* afficher largeur etoiles et un saut de ligne */
        for (j = 0; j < largeur; j = j + 1) /* pour chaque colonne */
        {
            printf("*");
        }
        printf("\n");
    }
}

int saisie_utilisateur()
{
    int n = 0;
    printf("entrer un entier : ");
    scanf("%d", &n);
    return n;
}

int binomial(int n, int p)
{
    return factorielle(n) / (factorielle(p) * factorielle(n - p));
}
```

---

4 Programmer avec des fonctions et procédures

Dans les exercices suivants, l'objectif est de mettre au point des programmes entiers en y apportant vos propres fonctions et procédures. La méthode pour y arriver : découper le programmes en tâches simples à mettre en œuvre et progresser par étapes en testant régulièrement votre code (même s'il est incomplet).

Pour chaque exercice, vous pouvez faire plus ou moins de fonctions et procédures selon si vous découpez le travail en de plus ou moins gros morceaux. Lorsqu'on découpe une tâche en beaucoup de morceaux on parle de *code ravioli*. Par exemple, pour afficher des motifs d'étoiles vous pouvez déclarer et définir les procédures suivantes :

```
/* afficher_ligne(5) affiche "*****\n" */
void afficher_ligne(int nb_etoiles);
/* afficher_etoile() affiche "*" */
void afficher_etoile();
/* afficher_espace() affiche " " */
void afficher_espace();
/* afficher_etoile() affiche "\n" */
void finir_ligne();
```

Lorsque c'est possible, exercez vous à produire du code ravioli en rajoutant des fonctions et procédures (par exemple, utiliser `afficher_ligne(n)` au lieu d'une boucle for pour afficher n étoiles et un retour à la ligne). Puis, réciproquement, à supprimer les sous-tâches qui vous paraissent exagérées (par exemple, remplacer un appel à `afficher_espace()` par un appel direct à `printf(" ")`.)

```c
#include <stdlib.h>
#include <stdio.h>

void afficher_triangle(int cote);

int main() {
    int i; /* var de boucle */

    /* testons différentes tailles de triangles */
    for (i = 2; i <= 10; i += 2) {
        printf("\n\nafficher_triangle(%d)\n", i);
        afficher_triangle(i);
    }
    return EXIT_SUCCESS;
}

void afficher_triangle(int cote) {
    int i; /* var de boucle */

    /* il faudrait deja arriver a faire un carre ! */
    for (i = 0; i < cote; i = i + 1) {/* pour chaque ligne */
        /* afficher une ligne de cote etoiles. Faire une sous-procedure ? */
        printf("*<-- %d -->*\n", cote);
    }
}
```

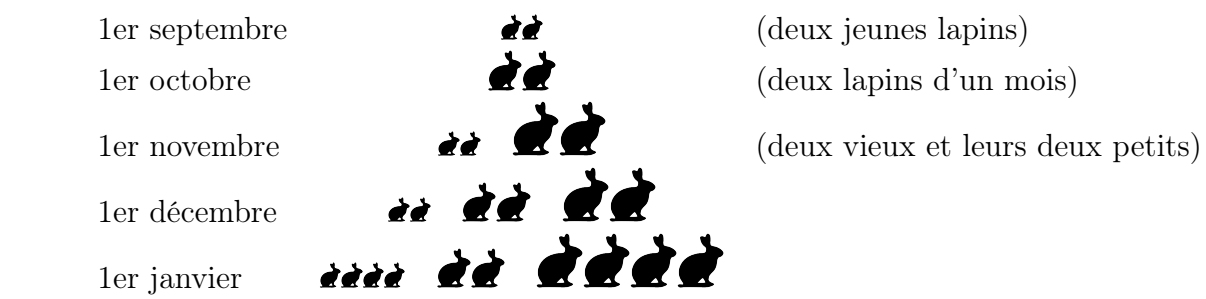
FIGURE 1 – Programme de Pippo pour mettre au point la fonction afficher triangle

4.1 Encore un triangle d'étoiles

Pippo a commencé à écrire un procédure qui permet d'afficher un triangle d'étoiles. Comme c'est un peu compliqué, il progresse par étapes et fait en sorte de pouvoir tester son code le plus tôt possible, même s'il n'est pas terminé. Bien entendu, le principal reste à faire. Vous pouvez récupérer ce programme sur la page web du cours. À vous de le compléter.

4.2 Population de lapins

Un couple de lapins a sa première portée à deux mois, puis une portée tous les mois. Chaque portée est un couple de lapins. Tous les couples ainsi obtenus se reproduisent de la même manière. [1]



1. On distinguera le nombre de couples de nouveaux lapins `nouveaux`, le nombre de couples de lapins ayant un mois, `un_mois`, et le nombre de couples de lapins ayant 2 mois ou plus, `vieux`. Calculer (à la main) le nombre de couples de lapins de chaque type, ainsi que leur nombre total, pour les 10 premiers mois.

1. Merci à Laure Petrucci pour cet exercice et à Lionel Allorge pour le dessin de lapin en cc-by-sa.

2. Écrire un algorithme `nb_lapins(nb_mois, nb_couples)` calculant le nombre de lapins obtenus au bout de `nb_mois` mois à partir de `nb_couples` couples jeunes, et renvoyant le résultat.
3. À combien s'élève la population au bout de 24 mois ?
4. Écrire un algorithme `lapins_un_milliard` calculant au bout de combien de temps les lapins sont plus d'un milliard (on supposera qu'aucun lapin ne meurt pendant cette période), en partant d'un couple de jeunes lapins.
5. Mettre en œuvre ces algorithmes dans un programme C.

Les lapins disparaissent ! Cinq couples de jeunes lapins se sont installé sur une île déserte et on fondé une communauté. Les premiers mois tout se passe normalement, mais à partir du septième mois, des lapins disparaissent après avori donné naissance à toutes les portées : exactement un cinquième des vieux lapins, quatre cinquièmes des lapins venant d'avoir deux mois et neuf dixièmes des nouveaux lapins survivent (par arrondi inférieur), tous les autres disparaissent. Ceci tous les mois, sans que la cause ne soit connue (certains scientifiques lapins mettent en cause la nourriture transgénique ou les antibiotiques large spectre, d'autres lapins évoquent une famille de renards, et d'autres encore pensent que les services secrets humains ont découvert le plan d'invasion de la Terre par les lapins et ont pris des contre-mesures).

6. Tenir compte de ces nouvelles données dans la simulation. Que se passe t'il ?
7. Afficher la population de lapin obtenue à chaque nouveau mois sous forme graphique sur une ligne. Par exemple, un couple de nouveaux lapins sera affiché par le caractère point "`.`", un couple de lapins d'un mois par le caractère "`o`", un vieux couple de lapins par le caractère "`#`".
8. Combien de lapins sont nés sur l'île ?

Correction. *wq*

```
/* lapins.c (I3) */
#include <stdlib.h> /* EXIT_SUCCESS */
#include <stdio.h>  /* printf */

#define DISPARITION 1

/* declaration de fonctions utilisateurs */
int simuler_population(int nb_mois, int nb_couples);
void afficher_population(int nouveaux, int un_mois, int vieux);
int lapins_un_milliard(int nb_couples);

int main()
{
#if DISPARITION == 0
    int n;
    /* nb_lapins après 24 mois */
    printf("nb_lapins apres 24 mois : %d\n", simuler_population(24, 1));
    /* depasser le milliard a partir d'un couple en */
    n = lapins_un_milliard(1);
    printf("apres %d mois il y plus d'un milliard de lapins, exactement : %d\n", n, simuler_
#else
    printf("nb_lapins apres 50 mois : %d\n", simuler_population(50, 5));
#endif

    /* valeur fonction */
```

```
    return EXIT_SUCCESS;
}

/* definitions des fonctions utilisateurs */
int simuler_population(int nb_mois, int nb_couples)
{
    int nouveaux = 2*nb_couples;
    int un_mois = 0;
    int vieux = 0;
    int naissances = 0;
#if DISPARITION
    int tot_naissances = 0;
#endif
    int i; /* var boucle */
    for (i = 1; i <= nb_mois; i = i + 1) {
/* les un_mois sont maintenant vieux (de deux_mois) */
vieux = vieux + un_mois;
/* il y a nb de vieux naissances */
naissances = vieux;  /* plutot naissances =  2 * (vieux / 2) */
/* les ex-nouveaux ont un mois */
un_mois = nouveaux;
/* les naissances font les nouveaux */
nouveaux = naissances;
#if DISPARITION
tot_naissances = tot_naissances + naissances;
if (i >= 7) {
    /* 4/5 des vieux meurent */
    vieux = vieux / 5;
    /* 2/5 des un_mois */
    un_mois = 4 * un_mois / 5;
    nouveaux = 9 * nouveaux / 10;
}
printf("%2d| ", i);
afficher_population(nouveaux, un_mois, vieux);
#endif
    }
    /* population de lapins après nb_mois */
#if DISPARITION
    printf("Nombre total de naissances : %d\n", tot_naissances);
#endif
    return vieux + un_mois + nouveaux;
}

void afficher_population(int nouveaux, int un_mois, int vieux)
{
/* ..... ooooo ##### */
    int i;
    for (i = 0; i < nouveaux / 2; i = i + 1)
    {
printf(".");
    }
    if (nouveaux > 0)
    {
printf(" ");
    }
    for (i = 0; i < un_mois / 2; i = i + 1)
```

```
    {
printf("o");
    }
    if (un_mois > 0)
    {
printf(" ");
    }
    for (i = 0; i < vieux / 2; i = i + 1)
    {
printf("#");
    }
    if (vieux > 0)
    {
printf(" ");
    }
    printf("%d\n", nouveaux + un_mois + vieux);
}

int lapins_un_milliard(int nb_couples)
{
/* one ne recode pas la simulation ce qui fait des calculs inutiles */
    int n = 0;
    /* dépasser le milliard */
    while (simuler_population(n, nb_couples) < 1000000000) {
n =n + 1;
    }

    return n;
}
```

`main()`

ligne	x	y	z	Affichage
initialisation	-3	5	?	
21				
21	3			
22				
22			3	
23				
23			6	
24				
24			5	
27	renvoie EXIT_SUCCESS			

`valeur_absolue(-3)`

ligne	n	Affichage
initialisation	-3	
35	renvoie 3	

`minimum(3, 5)`

ligne	a	b	Affichage
initialisation	3	5	
49	renvoie 3		

`factorielle(3)`

ligne	n	i	res	Affichage
initialisation	3	?	1	
63		1		
65			1	
66		2		
65			2	
66		3		
65			6	
66		4		
69	renvoie 6			

`minimum(5, 6)`

ligne	a	b	Affichage
initialisation	5	6	
49	renvoie 5		

TABLE 1 – Trace du programme de l'exercice 2.