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));
}
```

`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.