

Travaux dirigés 11 : struct, fonctions récursives

```
1  /* Declaration de fonctionnalites supplementaires */
2  #include <stdlib.h> /* EXIT_SUCCESS */
3  #include <stdio.h> /* printf() */
4
5  /* Declarations constantes et types utilisateurs */
6
7  struct duree_s
8  {
9      int h; /* heures */
10     int m; /* minutes */
11     int s; /* secondes */
12 };
13
14 /* Declarations de fonctions utilisateurs */
15
16 /* Multiplier une duree par un facteur (positif) */
17 struct duree_s multiplier_duree(int facteur, struct duree_s d);
18 /* Normaliser une duree (minutes et secondes seront dans [0, 59]) */
19 struct duree_s normaliser_duree(struct duree_s d);
20 /* Afficher une duree */
21 void afficher_duree(struct duree_s d);
22
23 /* Fonction principale */
24 int main()
25 {
26     /* Declaration et initialisation des variables */
27     struct duree_s duree_unitaire = {1, 32, 14}; /* Duree unitaire */
28     int seances = 100; /* Nombre de seances */
29     struct duree_s duree_totale; /* Duree totale */
30
31     /* Affichage */
32     printf("Duree d'une seance : ");
33     afficher_duree(duree_unitaire);
34     printf("\n");
35
36     duree_totale = multiplier_duree(seances, duree_unitaire);
37
38     /* Affichage */
39     printf("Duree totale des %d seances : ", seances);
40     afficher_duree(duree_totale);
41     printf("\n");
42
43     /* Valeur fonction */
44     return EXIT_SUCCESS;
45 }
46
47 /* Definitions de fonctions utilisateurs */
48 struct duree_s multiplier_duree(int facteur, struct duree_s d)
49 {
50     struct duree_s res; /* resultat */
51     /* Calcul */
52     res.h = d.h * facteur;
53     res.m = d.m * facteur;
```

```

54     res.s = d.s * facteur;
55     /* Normalisation */
56     res = normaliser_duree(res);
57     /* Retour valeur */
58     return res;
59 }
60
61 struct duree_s normaliser_duree(struct duree_s d)
62 {
63     struct duree_s res; /* resultat */
64     /* Normalisation secondes */
65     res.m = d.m + d.s / 60;
66     res.s = d.s % 60;
67     /* Normalisation minutes */
68     res.h = d.h + res.m / 60;
69     res.m = res.m % 60;
70     /* Retour valeur */
71     return res;
72 }
73
74 void afficher_duree(struct duree_s d)
75 {
76     printf("%d heures %d minutes %d secondes", d.h, d.m, d.s);
77 }

```

1 Enregistrements (struct)

1. Faire la trace du programme précédent.

Correction.

Trace. Voir le tableau 1 page 8.

2. Modifier le programme précédent (notamment le `struct duree_s`) de manière à ce que les durées intègrent un nombre de jours et que les heures soient dans l'intervalle $[0, 23]$.

Correction. Juste le diff avec le programme précédent.

```

8a9
>     int j; /* jours      */
27c28
<     struct duree_s duree_unitaire = {1, 32, 14}; /* Duree unitaire */
---
>     struct duree_s duree_unitaire = {0, 1, 32, 14}; /* Duree unitaire */
51a53
>     res.j = d.j * facteur;
69a72,74
>     /* Normalisation heures */
>     res.j = d.j + res.h / 24;
>     res.h = res.h % 24;
76c81
<     printf("%d heures %d minutes %d secondes", d.h, d.m, d.s);
---
>     printf("%d jours %d heures %d minutes %d secondes", d.j, d.h, d.m, d.s);

```

3. Écrire une fonction réalisant la somme de deux durées.

Correction. Direct. (On n'utilise pas ici de variable res pour le résultat : ce serait sans doute mieux, surtout que finalement on ne fait pas la trace de cette fonction).

```
struct duree_s somme_duree(struct duree_s d1, struct duree_s d2)
{
    /* Calcul */
    d1.s = d1.s + d2.s;
    d1.m = d1.m + d2.m;
    d1.h = d1.h + d2.h;
    d1.j = d1.j + d2.j;
    /* Normalisation */
    d1 = normaliser_duree(d1);
    /* Retour resultat */
    return d1;
}
```

4. Changer de main. Déclarer un tableau `durees` de 5 durées initialisé avec les durées : 1h30, 3h, 1h30, 3h, 15min; et un tableau d'entiers `seances` de taille 5 initialisé avec 12, 12, 12, 2, 1. Le tableau `durees` représente les durées des cours, TD, TP, partiels et colles de l'UE éléments d'informatique et le second tableau le nombre d'occurrences de ces séances durant le semestre. Compléter le main pour qu'il calcule le temps total consacrée à l'UE, hors révisions.

Correction. Le main devient :

```
int main()
{
    /* Declaration et initialisation des variables */
    struct duree_s durees[N] = /* durees des differentes seances */
    {
        {0, 1, 30, 0}, /* cours */
        {0, 3, 0, 0}, /* TD */
        {0, 1, 30, 0}, /* TP */
        {0, 3, 0, 0}, /* partiel */
        {0, 0, 15, 0} /* colles */
    };
    int seances[N] = {12, 12, 12, 2, 1}; /* nombre de seances */
    struct duree_s duree_totale = {0, 0, 0, 0}; /* Duree totale */
    int i; /* var de boucle */
    /* Calcul */
    for (i = 0; i < N; i = i + 1)
    {
        duree_totale = somme_duree(duree_totale,
                                   multiplier_duree(seances[i], durees[i]));
    }

    printf("Duree totale : ");
    afficher_duree(duree_totale);
    printf("\n");
    /* Valeur fonction */
    return EXIT_SUCCESS;
}
```

(Ce n'est pas demandé.) L'exécution donne l'affichage :

```
P0:TD11$ ./duree
Duree totale : 3 jours 6 heures 15 minutes 0 secondes
```

5. Comment faire en sorte que les données initiales (durées des séances, nombre de séances) soient renseignées dans un seul et même tableau, sans modifier `struct duree_s`?

Correction. Il faut créer un nouveau struct contenant une durée et un facteur entier.

```
struct cours_s
{
    int          n; /* nombre de seances */
    struct duree_s d; /* duree de chaque seance */
};
```

Le main devient :

```
int main()
{
    /* Declaration et initialisation des variables */
    struct cours_s ei[N] = /* durees des differentes seances */
    {
        {12, {0, 1, 30, 0}}, /* cours */
        {12, {0, 3, 0, 0}}, /* TD */
        {12, {0, 1, 30, 0}}, /* TP */
        {2, {0, 3, 0, 0}}, /* partiel */
        {1, {0, 0, 15, 0}} /* colles */
    };
    struct duree_s duree_totale = {0, 0, 0, 0}; /* Duree totale */
    int i; /* var de boucle */
    /* Calcul */
    for (i = 0; i < N; i = i + 1)
    {
        duree_totale = somme_duree(duree_totale,
                                   multiplier_duree(ei[i].n, ei[i].d));
    }

    printf("Duree totale : ");
    afficher_duree(duree_totale);
    printf("\n");
    /* Valeur fonction */
    return EXIT_SUCCESS;
}
```

Les struct ne sont pas si complexes et point difficiles

6. Proposer un type utilisateur pour les nombres complexes en notation algébrique ($a + ib$, $a, b \in \mathbb{R}$).

Correction.

```
struct complexe_s
{
    double re; /* partie reelle */
    double im; /* partie imaginaire */
}
```

7. Donner une fonction d'addition et une fonction de multiplication entre ces nombres complexes.

Correction.

```
struct complexe_s addition_complexe(struct complexe_s z1, struct complexe_s z2)
{
    z1.re = z1.re + z2.re;
    z1.im = z1.im + z2.im;
    return z1;
}
```

```

struct complexe_s multiplication_complexe(struct complexe_s z1, struct complexe_s z2)
{
    struct complexe_s res;
    res.re = z1.re * z2.re - z1.im * z2.im;
    res.im = z2.re * z1.im + z1.re * z2.im;
    return res;
}

```

8. Proposer un type utilisateur pour les points de l'espace (en coordonnées réelles).

Correction.

```

struct point_s
{
    double x; /* abscisses */
    double y; /* ordonnees */
    double z; /* hauteur */
}

```

9. Écrire une fonction `est_dans_sphere` prenant en argument un point c , un réel r , et un point p , qui renvoie vrai si le point p appartient à la sphère de centre c et de rayon r , faux sinon.

Correction.

```

est_dans_sphere(struct point_s c, double r, struct point_s p)
{
    return (p.x - c.x)*(p.x - c.x)
        + (p.y - c.y)*(p.y - c.y)
        + (p.z - c.z)*(p.z - c.z)
        <= r*r;
}

```

Remarque : on pourrait définir une fonction distance calculant la distance entre deux points, mais nous n'avons pas vu comment accéder au calcul de la racine carrée.

10. Quel pourrait être le type utilisateur pour une sphère ?

Correction. Une sphère est donnée par son centre (un point) et son rayon. Le type serait donc le suivant :

```

struct sphere_s
{
    struct point_s centre; /* centre */
    double         rayon; /* rayon */
}

```

2 Programmation récursive

1. Écrire la fonction factorielle de manière récursive, et faire la trace d'un appel à `factorielle(4)`.

Correction. Il vaut mieux, pour la trace prendre cette forme verbeuse (emploi de la variable `res`).

```

1  int factorielle(int n) /* factorielle recursive verbeuse */
2  {
3      int res = 1;
4      if (n > 1)
5      {
6          res = n * factorielle(n - 1); /* <--- appel recursif */
7      }
8      return res;
9  }

```

factorielle(4)

ligne	n	res	Affichage
initialisation	4	1	
6		24	
8	renvoie 24		

factorielle(3)

ligne	n	res	Affichage
initialisation	3	1	
6		6	
8	renvoie 6		

factorielle(2)

ligne	n	res	Affichage
initialisation	2	1	
6		2	
8	renvoie 2		

factorielle(1)

ligne	n	res	Affichage
initialisation	1	1	
8	renvoie 1		

- Écrire une fonction récursive pour le calcul du PGCD de deux entiers.

Correction.

« Étant donnés deux entiers naturels a et b , on commence par tester si b est nul. Si oui, alors le P.G.C.D. est égal à a . Sinon, on calcule c , le reste de la division de a par b . On remplace a par b , et b par c , et on recommence le procédé. Le dernier reste non nul est le P.G.C.D. » (wikipedia.fr, dans le passé).

Il faudra certainement aider la grande majorité des étudiants. En gros, il faut rappeler que le PGCD de deux entiers positifs a et b est défini par : $\text{PGCD}(a, b) = a$ si $b = 0$ et $\text{PGCD}(a, b) = \text{PGCD}(b, a \bmod b)$.

```

int pgcd(int a, int b)
{
    if (b == 0)
    {
        return a;
    }
    return pgcd(b, a % b);
}

```

Remarque : si l'entier le plus grand est b , la fonction échange ses arguments au premier appel récursif.

main()

ligne	duree_unitaire			seances	duree_totale			Affichage (sortie écran)
	h	m	s		h	m	s	
initialisation	1	32	14	100	?	?	?	
32								Duree d'une seance :
33								1 heures 32 minutes 14 secondes
34								<retour ligne>
36					153	43	20	
39								Duree d'une seance :
40								153 heures 43 minutes 20 secondes
41								<retour ligne>
44	SORTIE AVEC SUCCÈS							

afficher_duree({1, 32, 14})

ligne	d			Affichage
	h	m	s	
initialisation	1	32	14	
76				1 heures 32 minutes 14 secondes
77	ne renvoie rien			

multiplier_duree(100, {1, 32, 14})

ligne	facteur	d			res			Affichage
		h	m	s	h	m	s	
initialisation	100	1	32	14	?	?	?	
52					100			
53						3200		
54							1400	
56					153	43	20	
58	renvoie {153, 43, 20}							

normaliser_duree({100, 3200, 1400})

ligne	d			res			Affichage
	h	m	s	h	m	s	
initialisation	100	3200	1400	?	?	?	
65					3223		
66						20	
68				153			
69					43		
71	renvoie {153, 43, 20}						

afficher_duree({153, 43, 20})

ligne	d			Affichage
	h	m	s	
initialisation	153	43	20	
76				153 heures 43 minutes 20 secondes
77	ne renvoie rien			

TAB. 1 – Trace du programme de l'exercice 1.