Université Paris-Nord — L1
Institut Galilée — Année 2011-2012
Licence 1 - 1er semestre

Travaux dirigés 10 : types composées `struct`

```
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.
2. Modifier le programme précédent (notamment le `struct duree_s`) de manière à ce que les durées intégrent un nombre de jours et que les heures soient dans l'intervalle $[0, 23]$.
3. Écrire une fonction réalisant la somme de deux durées.
4. Changer de `main`. Déclarer un tableau `durees` de cinq durées initialisé avec les durées : 1h30, 3h, 1h30, 3h, 1h30 ; et un tableau `seances` de cinq entiers initialisé avec 12, 12, 12, 2, 1. Le tableau `durees` représente les durées des cours, TD, TP, partiels et prérentrée 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.
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` ?

Les `struct` ne sont pas si complexes et point difficiles

6. Proposer un type utilisateur pour les nombres complexes en notation algébrique ($a + \mathrm{i}b$, $a, b \in \mathbb{R}$).
7. Donner une fonction d'addition et une fonction de multiplication entre ces nombres complexes.
8. Proposer un type utilisateur pour les points de l'espace en coordonnées réelles : $\mathbb{R}^3$.
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.
10. Définir une fonction `distance` qui retourne la distance entre deux points de l'espace passés en arguments. Simplifier la fonction `est_dans_sphere` en faisant appel à cette fonction distance.
11. Proposer un type utilisateur pour représenter une sphère dans l'espace.
12. Définir une fonction `collision_spheres` qui prend en entrée deux sphères et retourne `TRUE` si les deux sphères ont une intersection non vide, `FALSE` sinon.