Informatique

Sequence 6 : Types structurés

TP 6.1

3h - v1

IUT d'Annecy - Département GEII - 9 Rue de l'Arc en Ciel, 74940 Annecy

Introduction aux types structurés

Objectifs du TP

Ce TP vous permet de découvrir et manipuler les **types structurés** (structures) en langage C. Les structures sont un concept fondamental qui permet de regrouper plusieurs données de types différents en une seule entité logique.

Compétences visées :

- Déclarer et utiliser des structures simples
- Manipuler des tableaux de structures
- Créer des structures imbriquées
- Passer des structures en paramètres de fonctions
- Résoudre des problèmes réalistes avec les structures

Validation des exercices

Les premiers exercices peuvent être validés avec l'outil `check50` :

```
check50 IUT-GEII-Annecy/exercices/2025/info1/tp_structures/nom_exercice
```

Conseils :

- Testez votre code manuellement avant d'utiliser `check50`
- Lisez attentivement les messages d'erreur
- N'hésitez pas à utiliser le débogueur (`gdb` ou l'extension VS Code)
- Demandez de l'aide à votre enseignant si vous êtes bloqué

Conseils méthodologiques

Pour réussir ce TP :

1. **Lisez la théorie** avant de commencer les exercices
2. **Commencez simple** : faites les premiers exercices même s'ils semblent faciles
3. **Testez progressivement** : compilez et testez après chaque modification
4. **Utilisez des noms explicites** : nommez vos variables et fonctions clairement
5. **Commentez votre code** : cela vous aidera à comprendre votre propre logique

Durée estimée : 3h

Bon travail !

Un peu de théorie

Les types structurés en C

Un **type structuré** (ou `struct`) permet de regrouper plusieurs données de types différents en une seule entité logique.

Avantages :

- Regrouper des données liées ensemble (cohésion)
- Créer des types de données personnalisés
- Améliorer la lisibilité du code
- Faciliter la manipulation de données complexes

Syntaxe de déclaration :

```
struct NomStructure {
    type1 champ1;
    type2 champ2;
    // ...
};
```

Exemple concret :

```
struct Point {
    float x;
    float y;
};
```

Utilisation d'une structure

Création d'une variable de type structure :

```
struct Point p1;  // Déclare une variable p1 de type struct Point
```

Accès aux champs :

```
p1.x = 3.5;       // Affecte 3.5 au champ x
p1.y = 2.7;       // Affecte 2.7 au champ y
printf("Point: (%.1f, %.1f)\n", p1.x, p1.y);
```

Initialisation lors de la déclaration :

```
struct Point p2 = {1.0, 2.0};  // x=1.0, y=2.0
```

Typedef : simplifier l'utilisation

Le mot-clé `typedef` permet de créer un alias pour une structure, simplifiant son utilisation.

Sans typedef :

```
struct Point {
    float x;
    float y;
};
struct Point p1;  // On doit écrire "struct Point"
```

Avec typedef :

```
typedef struct {
```

```
    float x;
    float y;
} Point;

Point p1;  // Plus simple ! Pas besoin de "struct"
```

Structures et fonctions

Les structures peuvent être passées en paramètres et retournées par des fonctions.

Passage par valeur (copie) :

```
void afficher_point(Point p) {
    printf("(%.1f, %.1f)\n", p.x, p.y);
}
```

Passage par adresse (modification possible) :

```
void deplacer_point(Point *p, float dx, float dy) {
    p->x += dx;  // Notation -> pour accéder aux champs via pointeur
    p->y += dy;
}
```

Retour de structure :

```
Point creer_point(float x, float y) {
    Point p;
    p.x = x;
    p.y = y;
    return p;
}
```

1 Premiers pas avec les structures

Vous pouvez télécharger un squelette sur le lien suivant :

```
wget https://github.com/IUT-GEII-Annecy/squelettes/releases/download/branch-2025/tp6.zip
```

Dans cette première partie, vous allez découvrir les structures en manipulant des points dans un plan 2D.

1.1 Déclaration et utilisation basique

Manipulation 1 : Déclarer et afficher un point

Objectif : Créer une structure Point et l'utiliser.
Cahier des charges :

- Déclarer une structure `Point` avec deux champs `float x` et `float y`
- Dans le `main`, créer une variable `p1` de type `Point`
- Demander à l'utilisateur les coordonnées x et y
- Afficher le point sous la forme : `Point: (x, y)`

Exemple d'exécution :

```
Entrez x: 3.5
Entrez y: 2.8
Point: (3.5, 2.8)
```

```
check50 IUT-GEII-Annecy/exercices/2025/info1/tp_structures/point_simple
```

Manipulation 2 : Fonction de création de point

Objectif : Créer une fonction qui retourne une structure.
Cahier des charges :

- Écrire une fonction `Point creer_point(float x, float y)` qui crée et retourne un `Point`
- Écrire une fonction `void afficher_point(Point p)` qui affiche un point sous la forme `(x, y)`
- Dans le `main`, utiliser ces fonctions pour créer et afficher 3 points

Manipulation 3 : Calculer la distance à l'origine

Contexte : On souhaite calculer la distance entre un point et l'origine (0, 0).
Formule : $d = \sqrt{x^2 + y^2}$ (utiliser `sqrt` de `math.h`)
Cahier des charges :

- Reprendre la structure `Point`
- Écrire une fonction `float distance_origine(Point p)` qui calcule et retourne la distance
- Dans le `main`, créer un point, demander ses coordonnées et afficher sa distance à l'origine
- Arrondir l'affichage à 2 décimales

Note : Pour compiler avec `math.h`, ajouter `-lm` : `gcc fichier.c -lm`

```
check50 IUT-GEII-Annecy/exercices/2025/info1/tp_structures/distance_origine
```

1.2 Fonctions sur les structures

Manipulation 4 : Distance entre deux points

Contexte : Calculer la distance entre deux points quelconques.
Formule : $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$
Cahier des charges :

- Écrire une fonction `float distance(Point p1, Point p2)`
- Dans le `main`, demander les coordonnées de deux points
- Afficher la distance entre ces deux points (2 décimales)

Exemple :

```
Point 1 - x: 0
Point 1 - y: 0
Point 2 - x: 3
Point 2 - y: 4
Distance: 5.00
```

```
check50 IUT-GEII-Annecy/exercices/2025/info1/tp_structures/distance_deux_points
```

2 Tableaux de structures

Cette partie explore l'utilisation de tableaux de structures, une combinaison puissante pour gérer des collections de données complexes.

2.1 Premiers tableaux de structures

Manipulation 5 : Polygone — Calculer le périmètre

Contexte : Un polygone est défini par une séquence de points. On veut calculer son périmètre.
Cahier des charges :

- Utiliser la structure `Point` (x, y)
- Demander le nombre de sommets **n** ($3 \leq n \leq 20$)
- Lire les coordonnées de **n** points dans un tableau `Point sommets[20]`
- Calculer le périmètre : somme des distances entre points consécutifs
- Ne pas fermer le polygone (pas de distance entre dernier et premier point)
- Afficher le périmètre avec 2 décimales

Exemple :

```
Nombre de sommets: 4
Point 0: (0, 0)
Point 1: (1, 0)
Point 2: (1, 1)
Point 3: (0, 1)
Perimetre: 3.00
```

Manipulation 6 : Trouver le point le plus proche de l'origine

Contexte : Parmi un ensemble de points, identifier celui qui est le plus proche de l'origine.
Cahier des charges :

- Demander **n** ($1 \leq n \leq 50$) puis lire **n** points
- Trouver le point ayant la distance minimale à l'origine
- Afficher l'indice de ce point (0..n-1) et sa distance à l'origine (2 décimales)
- En cas d'égalité, afficher le premier trouvé

Exemple :

```
Nombre de points: 3
Point 0: (5, 5)
Point 1: (1, 1)
Point 2: (2, 3)
Point le plus proche: indice 1, distance 1.41
```

3 Structures imbriquées

Cette partie introduit les structures imbriquées et des applications plus réalistes.

3.1 Structures contenant d'autres structures

Manipulation 7 : Rectangle — Calcul de l'aire

Contexte : Un rectangle est défini par deux points : coin supérieur gauche et coin inférieur droit.
Structures à utiliser :

```c
typedef struct {
    float x;
    float y;
} Point;

typedef struct {
    Point coin_haut_gauche;
    Point coin_bas_droit;
} Rectangle;
```

Cahier des charges :

- Demander les coordonnées des deux coins du rectangle
- Calculer l'aire : largeur × hauteur
- Largeur $= |x_2 - x_1|$, Hauteur $= |y_2 - y_1|$
- Afficher l'aire avec 2 décimales

Exemple :

```
Coin haut-gauche - x: 0
Coin haut-gauche - y: 3
Coin bas-droit - x: 4
Coin bas-droit - y: 0
Aire: 12.00
```

Manipulation 8 : Cercle — Intersection avec un point

Contexte : Vérifier si un point est à l'intérieur d'un cercle.
Cahier des charges :

- Proposer une structure pour représenter un cercle.
- Demander le centre du cercle (x, y) et son rayon
- Demander les coordonnées d'un point à tester
- Calculer la distance entre le point et le centre
- Afficher `INTERIEUR` si distance $\leq$ rayon, sinon `EXTERIEUR`

Exemple :

```
Centre du cercle - x: 0
Centre du cercle - y: 0
Rayon: 5
Point a tester - x: 3
Point a tester - y: 4
INTERIEUR
```

Note : Distance = 5, exactement sur le cercle, donc INTERIEUR.