Types opaques

Programmation séquentielle en C, 2024-2025

Orestis Malaspinas (A401)

2025-02-28

Informatique et Systèmes de Communication, HEPIA

Types composés

- Jusqu'ici les `struct` sont dans les `.h` et sont *transparents*

```
// table.h
typedef struct _table {
    int *data;
    int length;
} table;
// main.c
table tab; // membres de tab accessibles directement
tab.length = 10;
tab.data = malloc(tab.length * sizeof(int));
tab.data[9] = 10;
```

Types opaques

- Afin de cacher les détails de l'implémentation.
- Afin d'éviter les modifications directs des données.
- Afin de protéger le monde de la dévastation!
- Définition de types **opaques**:
  - Variables dans les structures ne sont pas accessibles.
  - Variables dans les structures ne sont pas modifiables.
  - Les variables ne sont même pas connues.
- Nécessité de passer par des fonctions pour initialiser/modifier les instances de types opaques.
- Très souvent utilisés pour les structures de données abstraites (table de hachage, pile, file, ...).

Utilisation d'un type opaque: problème?

- Dans `opaque.h`

```
struct table;
```

- Dans `opaque.c`

```
struct table {
    int a;
}
```

- Dans `main.c`

```
int main() {
    struct table t;
}
// error: storage size of ''t 'isnt known
```

- La taille de `table` n'est pas connue à la compilation!
- Comment faire?

Utilisation d'un type opaque: pointeur!

- Dans `opaque.h`

```
struct table;
struct table *create();
void init(struct table **t);
```

- Dans `opaque.c`

```
struct table {
    int a;
}
struct table *create() {
    struct table *t = malloc(sizeof(*t));
    return t;
}
void init(struct table **t) {
    *t = malloc(sizeof(**t));
}
```

- Dans `main.c`

```
int main() {
    struct table *t = create();
    init(&t);
    t->a = 2; // Interdit, set(2)
    printf("%d\n", t->a); // Interdit, get()
}
```

Un peu plus joli: typedef! (1/2)

- Dans `opaque.h`

```
struct _table;
typedef struct _table * table;
void init(table *t);
void set_a(table t, int a);
int get_a(table t);
```

- Dans `opaque.c`

```
struct _table {
    int a;
}
void init(table *t) {
    *t = malloc(sizeof(**t));
    (*t)->a = 0;
}
void set_a(table t, int a) {
    t->a = a;
}
int get_a(table t) {
    return t->a;
}
```

Un peu plus joli: typedef! (2/2)

- Dans `main.c`

```c
int main() {
    table t;
    init(&t);
    set_a(t, 10);
    printf("%d\n", get_a(t));
}
```

- On a fait les fonctions `get_a()` et `set_a()` comme exemples, mais...

Un peu plus joli: typedef! (2/2)

- Dans `main.c`

```
int main() {
    table t;
    init(&t);
    set_a(t, 10);
    printf("%d\n", get_a(t));
}
```

- On a fait les fonctions `get_a()` et `set_a()` comme exemples, mais...
- c'est pas forcément nécessaire d'implémenter (`get/set`).

Un peu plus joli: typedef! (2/2)

- Dans `main.c`

```
int main() {
    table t;
    init(&t);
    set_a(t, 10);
    printf("%d\n", get_a(t));
}
```

- On a fait les fonctions `get_a()` et `set_a()` comme exemples, mais...
- c'est pas forcément nécessaire d'implémenter (`get/set`).
- Par exemple, pour la hashmap on `get/set` les variables des structs!

Yaka

- Utiliser les types opaques pour la hashmap!