COMPLÉMENT

STRUCTURES & POINTEURS DE FONCTION

Vincent Aranega

POINTER DE FONCTION DANS UNE STRUCTURE?

- Comme les pointeurs de fonctions sont… des pointeurs, il est possible de les ajouter comme champ dans une structure

Interêt?

- "Centralisation" d'une fonction par rapport au type
- Donne un semblant (mais vraiment un semblant) de programmation objet

"PROGRAMMATION ORIENTÉ OBJET" EN C

- La programmation orienté objet n'existe pas en C
- Il manque à C:
  - une vrai notion de classe
  - un moyen d'accéder à l'instance de classe (`self` ou `this`)
  - l'héritage
  - un opérateur de création d'instance

"PROGRAMMATION ORIENTÉ OBJET" EN C

- On peu tricher
  - notion de classe → structure
  - moyen d'accéder à l'instance de classe → passage en paramètres
  - l'héritage → Nope, on fait pas ici (mais c'est possible de tricher)
  - un opérateur de création d'instance → fonction de création

UNE "CLASSE" EN C

- Chaque structure définit devra comporter au moins
  - Un constructeur (init des champs)
    → ptr de fonction
  - Un destructeur (free des allocations potentielles)
    → ptr de fonction
- Toute méthode supplémentaire sera ajouté à la structure
  → ptr de fonction
- Les attributs sont des champs "classique" de structure

UNE "CLASSE" EN C: EXEMPLE

Définition de la classe

```
typedef struct Travailleur {
    char *nom;
    char *prenom;
    int salaire;

    void (*destroy)(struct Travailleur*);
    void (*credite_salaire)(struct Travailleur*, int);
    void (*affiche_salaire)(struct Travailleur*);
} Travailleur;
```

UNE "CLASSE" EN C: FONCTION D'INIT/FREE

```c
void Travailleur_init(Travailleur* this, char* prenom, char* nom)
{
    this->prenom = (char *)malloc(strlen(prenom) * sizeof(char) + 1);
    this->nom = (char *)malloc(strlen(nom) * sizeof(char) + 1);
    strcpy(this->prenom, prenom);
    strcpy(this->nom, nom);
    this->salaire = 0;
}

void Travailleur_Free(Travailleur* this)
{
    free(this->prenom);
    free(this->nom);
    free(this);
}
```

UNE "CLASS EN C": MÉTHODES

```c
void Travailleur_credite_salaire(Travailleur* this, int somme)
{
    this->salaire += somme;
}

void Travailleur_affiche_salaire(Travailleur* this)
{
    printf("%s %s: %d\n", this->prenom, this->nom, this->salaire);
}
```

UNE "CLASSE" EN C: CRÉATION D'INSTANCES

```
Travailleur* Travailleur_New(char* prenom, char* nom)
{
    Travailleur* this = (Travailleur *)malloc(sizeof(Travailleur));
    // On init la nouvelle "instance"
    Travailleur_init(this, prenom, nom);
    // On "attache" les fonctions à l'instance créée
    this->destroy = &Travailleur_Free;
    this->credite_salaire = &Travailleur_credite_salaire;
    this->affiche_salaire = &Travailleur_affiche_salaire;
    return this;
}
```

UNE "CLASSE" EN C: MAIN

```c
int main(void)
{
    Travailleur *t1 = Travailleur_New( "John", "Doe");
    t1->affiche_salaire(t1);
    t1->credite_salaire(t1, 1000);
    t1->affiche_salaire(t1);
    t1->destroy(t1);
    return 0;
}
```

CONCLUSION

- Il est possible de "bricoler" une programmation qui ressemble à de l'objet
- Repose sur une utilisation intensive des pointeurs de fonction
- **Mais** les principes objets sont manquant
  - Nécessite le passage en paramètre de l'instance sur laquelle travailler
  - Obligation de la création de l'équivalent d'un `new`
  - …
- Donc possible en théorie, mais non recommandé en pratique