SORBONNE UNIVERSITÉ

TD : Programmation en C — semaine 6

23 mars 2018

Objectif(s)

- ✯ Introduction aux types structurés en C (`struct`).
- ✯ Méthodologie de développement logiciel par itération.

— **Note 1** : Pour tous vos programmes, vous utiliserez le script *run.sh* pour compiler.
— **Note 2** : Pour tous vos programmes, vous ferez des fonctions de 5 lignes maximum (sans comptez les commentaires, accolades seules ou en-têtes de fonctions).

Exercice 1 – La limite des tableaux

1. Écrire un programme qui demande à l'utilisateur d'entrer les prénoms et âges des membres de sa famille.
2. Partez d'une fonction de tri (par exemple à bulles) implémentée au cours des TDs précédents, et trier les membres de la famille suivant leurs âges. Vous afficherez ensuite leurs noms et âges triés. Il faut modifier la fonction de tri pour qu'elle échange également les prénoms dans le tableau correspondant.

Exercice 2 – En route vers les structures

Comme on vient de le voir, créer deux tableaux pour stocker les prénoms et âges n'est pas pratique du tout. D'un point de vue conceptuel, on voudrait pouvoir regrouper le prénom et l'âge d'une personne dans un concept "Personne". Le problème est que les tableaux ne permettent que de stocker des éléments qui ont le même type. On peut utiliser une autre structure permettant de stocker des types distincts :

```
struct Personne {
  char prenom[PRENOM_MAX];
  int age;
};
```

On peut utiliser cette structure de la manière suivante :

```
struct Personne p;
fgets(p.prenom, sizeof(p.prenom), stdin);
scanf("%d", &p.age);
```

On accède aux champs de la structure avec l'opérateur point `s.champs`. Si la structure est un pointeur, on peut accéder aux champs de la manière suivante :

```
void saisir_age(struct Personne *p) {
  scanf("%d", &((*p).age));
}
```

On déréférence d'abord le pointeur `p` et puis on accède au champ pointé. Il y a un raccourci avec l'opérateur flèche `s->champs` pour faire ce travail :

```
void saisir_age(struct Personne *p) {
  scanf("%d", &p->age);
}
```

1. Étudier le programme `personne.c` donné sur le site du cours.
2. À partir du programme `personne.c`, refaire l'exercice 1 avec un tableau de structures : `struct Personne famille[MAX_FAMILLE]` ou `struct Personne* famille` suivant l'allocation du tableau. Notez bien : les règles établies avec les tableaux restent exactement les mêmes avec les structures (un tableau pointe toujours vers son premier élément).

Exercice 3 – Dessine moi la mémoire

On utilise la structure `Personne` précédente pour ces diverses questions. Vous ferez les exercices suivants **en dessinant sur une feuille de papier** la mémoire des programmes suivants.

1. ```
   struct Personne p;
   ```

2. ```
   struct Personne p;
   p.prenom[0] = 'a';
   p.prenom[1] = '\0';
   p.age = 22;
   ```

3. Soit la fonction :

   ```
   void gerer_age(struct Personne p) {
       p.age = 10;
       printf("%d", p.age);
   }
   ```

   Dessinez la mémoire du programme suivant (y compris l'argument de la fonction "p").

   ```
   struct Personne p;
   p.prenom[0] = 'a';
   p.prenom[1] = '\0';
   p.age = 22;
   gerer_age(p);
   printf("%d", p.age);
   ```

   Qu'affichera t'il ?

4. Soit la fonction :

   ```
   void gerer_age(struct Personne* p) {
       p->age = 10;
       printf("%d", p->age);
   }
   ```

   Dessinez la mémoire du programme suivant (y compris l'argument de la fonction "p").

   ```
   struct Personne p;
   p.prenom[0] = 'a';
   p.prenom[1] = '\0';
   p.age = 22;
   gerer_age(&p);
   printf("%d", p.age);
   ```

   Qu'affichera t'il ?

5. Soit la fonction :

```
void gerer_prenom(struct Personne p) {
    p.prenom[0] = 'b';
    printf("%s", p.prenom);
}
```

Dessinez la mémoire du programme suivant (y compris l'argument de la fonction "p").

```
struct Personne p;
p.prenom[0] = 'a';
p.prenom[1] = '\0';
p.age = 22;
gerer_prenom(p);
printf("%s", p.prenom);
```

Qu'affichera t'il ?

6. Que se passe t'il si on utilise `gerer_prenom(struct Personne* p)` (avec les bons changements dans le programme) ?

7.
```
struct Personne famille[2];
```

8.
```
struct Personne* famille = malloc(2 * sizeof(struct Personne));
```

9.
```
void gerer_personne(struct Personne* p) {
    p->age=10;
}
int main() {
    struct Personne* f = (struct Personne*) malloc(2 * sizeof(struct Personne));
    f[0].age = 11;
    f[1].age = 12;
    gerer_personne(f);
    printf("%d %d", f[0].age, f[1].age);
    free(f);
    return 0;
}
```

Exercice 4 – Jeu de la bataille et développement itératif

Convention : On mettra toujours une majuscule aux noms de structures.

Le but de cet exercice est de créer une variante du jeu de carte de la bataille. Les règles sont les suivantes :
— Dans ce jeu, chaque joueur retourne une carte en même temps et la carte qui a la plus grande valeur gagne.
— Le joueur qui a gagné récupère la carte du perdant et la met dans son tas.
— Le joueur qui n'a plus de carte perd.
— En cas d'égalité, les joueurs remettent leur carte dans leur tas.
— Le gagnant peut décider de remettre sa carte dans son tas ou de la garder pour le tour suivant.
Un scénario du jeu finalisé sera comme suit :

```
Bienvenue sur le jeu de la bataille !

Veuillez entrer le nom du joueur 1 : Jean-mi
Veuillez entrer le nom du joueur 2 : Jean-pa
```

```
Les cartes ont été distribuée !
Démarrage de la partie.

"Valet de pique"  VS  "7 de cœur"
    Voulez-vous conserver votre "Valet de pique" Jean-mi [y/n] ? y

"Valet de pique"  VS  "As de cœur"
    Voulez-vous conserver votre "As de cœur" Jean-pa [y/n] ? n

"1 de carreau"  VS  "1 de trèfle"
    Remise dans le tas.

"3 de coeur"  VS  "4 de trèfle"
    Voulez-vous conserver votre "4 de trèfle" Jean-pa [y/n] ? n
...

On applaudit Jean-mi qui remporte la partie.
```

Nous allons décomposer la programmation de ce jeu en étapes successives.

Lorsqu'on commence à programmer des logiciels qui sont plus conséquent, il ne faut pas partir tête baissée dans le code. La première étape est de réfléchir au problème et comment on va l'aborder. Mais comment fait-on pour y réfléchir ? Deux aspects sont à prendre en compte :

— Quelles sont les structures de données (`struct`, tableaux, ...) dont on aura besoin ?
— Quelles seront les fonctions dont on aura besoin pour réaliser ce jeu.

Le point le plus **fondamentale** est de **découper son problème** en sous-problèmes plus faciles à résoudre.

1. Décrivez (sur papier !) les structures que vous pensez utiliser pour développer ce jeu. Il n'est pas nécessaire que ses structure soient définitives, elles sont là pour nous donner une idée de ce qu'on va manipuler.
2. Dans un fichier `iteration.txt`, découper le problème initial en 10 itérations initiales.
   — Pour chaque itération vous décrivez en une ligne la fonctionnalité à ajouter au logiciel courant.
   — Pensez qu'une itération doit vous prendre 10 minutes en terme de code.
   — Chaque itération propose un programme fonctionnel que vous pouvez lancer et exécuter (même si ce programme n'a pas toutes ses fonctionnalités).
   — Il faut donc tester que le programme marche bien entre chaque itération.

   Par exemple pour le projet du "Plus ou moins" :
   1. Afficher les messages de bienvenue et de sortie.
   2. Générer un nombre aléatoire et l'afficher à l'écran.
   3. Demander à l'utilisateur un nombre et l'afficher.
   4. Comparer les deux nombres et afficher un message correspondant au résultat du jeu.
   5. Créer une boucle de jeu qui termine quand l'utilisateur a trouvé le nombre.

   Pour pouvoir tester son logiciel, l'idée est d'afficher des résultats intermédiaires (par exemple : le nombre aléatoire ou celui de l'utilisateur) même si ceux-ci ne font pas partie du logiciel final.
3. **Les itérations sont flexibles !** Si vous vous rendez compte qu'une itération est trop complexe, divisez-la en sous-itérations et mettez à jour votre fichier `iteration.txt` (donc vous pouvez avoir plus que 10 itérations !). Pour chacune de ces itérations, vous créerez un programme `bataille-n.c` où n est remplacé par l'itération courante, par exemple `bataille-3.c` pour la troisième itération. Quand vous avez testé et vérifié qu'un programme à l'itération 3 marche, vous copiez ce fichier vers `bataille-4.c` et commencez l'itération 4.