CSH : Initiation au C et au shell

Première année

TD 2 : Encore les pointeurs . . .

★ **Exercice 1.** Un programme C contient la déclaration suivante :

```
char *couleur[6] = {"rouge", "vert", "bleu", "blanc", "noir","jaune"};
```

▷ **Question 1.** Que représente `couleur` ?

Réponse.

C'est un tableau de 6 pointeurs vers des chaines de caractères. Un schéma au tableau permet de mieux expliquer cela et ce qui suit . . .

▷ **Question 2.** Que désigne `couleur + 2` ?

▷ **Question 3.** Quelle est la valeur de `*couleur` ?

▷ **Question 4.** Quelle est la valeur de `*couleur + 2` ?

Réponse.

```
couleur + 2 = &couleur[2] et *(couleur+2)=couleur[2]=''bleu''
*couleur = couleur [0] = ''rouge''
*couleur+2 = ''uge''

*couleur pointe sur le premier élément de la chaine ''rouge'' soit
 'r'. Si on rajoute 2 (2 X sizeof(char)), ça pointera sur 'u'. La
 fin de la chaine est après 'e' de ''rouge''.
```

▷ **Question 5.** Quelle est la valeur de `*(*(couleur + 5) +3)` ? (Même question avec +4, +5, +6).

Réponse.

```
*(*(couleur + 5) +3) = 'n'
En effet :
*(couleur + 5) = 'j' , premier caractère de ''jaune'' situé à couleur[5]
*(couleur + 5) +3 = ''ne''
```

★ **Exercice 2. Pointeurs et structures.**

Soit le type suivant :

```
typedef struct {
  char nomPlanete[MAX_CAR];
  int  rayon;
} Planete;
```

▷ **Question 1.** Quelle est l'instruction qui permet de créer une variable *pointeur sur Planete* ?

Réponse.

```
Planete *p1;
p1 = (Planete *) malloc (sizeof(Planete));
```

▷ **Question 2.** Ecrire une fonction qui saisit les données relatives à une planète.

Réponse.

```
void Saisir(Planete *p){

  printf("Donner nom planète : ");
  scanf ("%s",p->nomPlanete);
  printf("Donner son rayon : ");
  scanf ("%d",&p->rayon);
}
```

▷ **Question 3.** Ecrire une fonction qui duplique une planète.

Réponse.

```
Planete* Dupliquer(Planete p){

  Planete *p1;
  p1 = (Planete *) malloc (sizeof(Planete));

  strcpy(p1->nomPlanete, p.nomPlanete);
  p1->rayon = p.rayon;

  return (p1);
}
```

▷ **Question 4.** Soit l'extrait suivant permettant de tester les fonctions précédentes ;

```
void main()
{
  Planete p;
  Planete *ptr_p;
  Saisir(&p);
  printf("%s %d \n", p.nomPlanete, p.rayon);
  ptr_p = Dupliquer(p);
  printf("%s %d \n", ptr_p->nomPlanete, ptr_p->rayon);
}
```

Comment s'analyse les types des différentes références suivantes :

Référence	Type
`ptr_p`	
`*ptr_p`	
`ptr_p->nomPlanete`	
`ptr_p->rayon`	
`p`	
`&p`	
`p.nomPlanete`	
`p.rayon`	

Réponse.

Référence	Type
`ptr_p`	Planete *
`*ptr_p`	Planete
`ptr_p->nomPlanete`	char []
`ptr_p->rayon`	int
`p`	Planete
`&p`	Planete*
`p.nomPlanete`	char []
`p.rayon`	int

★ **Exercice 3. Chaînes de caractères.**

Ecrire une fonction `PremierCar(...)` qui renvoie l'adresse de la première occurence du caractère `c` dans une chaîne de caractères dont l'adresse (du premier caractère) est passé à l'argument `ptrCar` :

```
char *PremierCar(char c, char *ptrCar)
```

Ecrire la fonction `int main(int argc, char *argv[])` permettant de tester la fonction. Un exemple d'exécution est :

```
./occurence  o Bonjour
```

Le résultat affiché est le suivant :

```
La premiere occurence de 'o' dans 'Bonjour' est en position 1
```

Réponse.

Cet exercice a été déjà fait en TP2 mais il y a une différence dans le type de retour qui est un pointeur au lieu d'un entier. Aussi, on fait usage des arguments en ligne de commande.

```
#include <stdio.h>

char *PremierCar(char c, char *ptrCar){
  do
    if (*ptrCar == c)
      return (ptrCar);
  while (*ptrCar++);

  return (NULL);
}

int main(int argc, char * argv[]){

  char c, *str, *adrPremCar;

  c = argv[1][0]; //argv[1] étant un pointeur vers une chaine de car.
                  // argv[1][0] est le premier car de cette chaine
  str = argv[2];

  if ( (adrPremCar = PremierCar(c,str)) == NULL)
    printf ("'%c' n'est pas dans %s.\n",c,str);
  else
    printf ("La premiere occurence de '%c' dans %s est en position
    %d.\n", c,str,adrPremCar - str);
}
```

★ **Exercice 4. Filiation.**

Sous Unix, il existe un ensemble de pointeurs représentant la filiation des processus :

Dans la figure 1, `fils A` est le premier processus fils créé par le processus père, `fils B` est le deuxième fils créé par le processus père . . .etc. Chaque fils peut évidemment créer des fils, et le père est également le fils d'un autre processus.

Nous supposons pour simplifier que chaque processus est caractérisé par un numéro (entier positif).

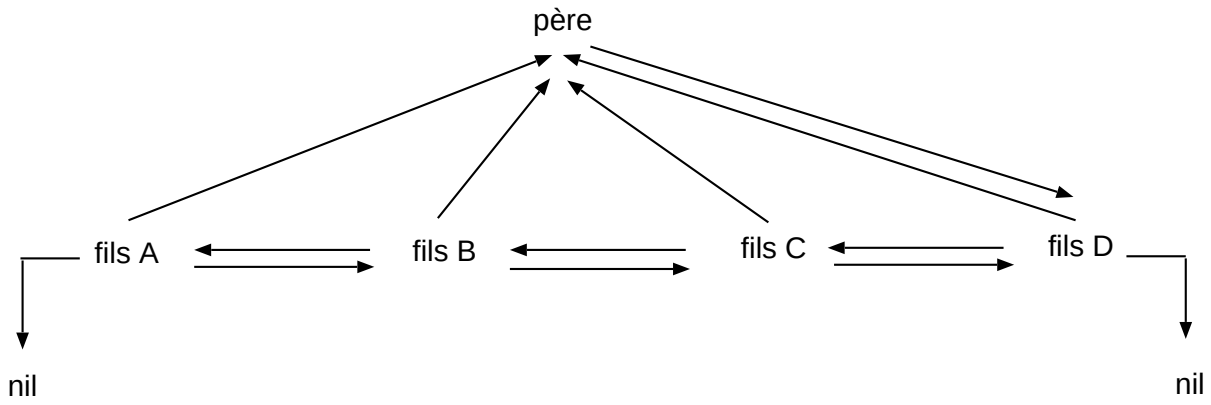

FIGURE 1 – *Filiation sous Unix*

▷ **Question 1.** Représenter shématiquement, sous la forme d'une structure de données, un nœud (c'est à dire un processus) de cette filiation, puis écrire la définition de cette structure en langage C.

▷ **Question 2.** Le processus repéré par la variable pointeur `p` vient de créer un processus fils de numéro `no`. Ecrire en langage C la fonction `maj` qui met à jour la filiation des processus.

Réponse.

```
struct proc{
  unsigned int noproc;
  struct proc *pere, *frere_g, *frere_d, *dernier_fils;

};

void maj(struct proc *p, int no){
  struct proc *fils;
  fils = (struct proc *) malloc ((unsigned int)(sizeof(struct proc)));

  fils->noproc = no;
  fils->pere = p;
  fils->frere_d = NULL;
  fils->dernier_fils = NULL

  if (p->dernier_fils != NULL){
    //le père a déjà des fils, le dernier fils devient le frère à
    //gauche et aura un frère à sa droite
    fils->frere_g = p->dernier_fils;
    p->dernier_fils->frere_d = fils;
  } else {// le père n'avait pas de fils
    fils->frere_g = NULL;
    p->dernier_fils = fils;
  }
}
```