TD2 : Encore des pointeurs...

CSH : Initiation au C et au shell

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

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

char **pcoul = couleur;
```

▷ **Question 1:** Que représente `couleur` ? Et `pcoul` ?

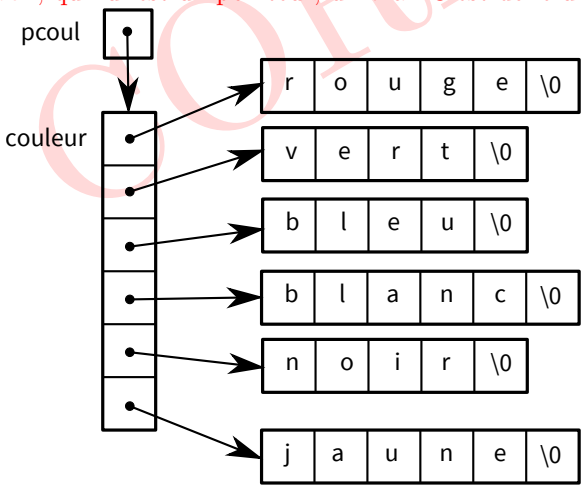
Réponse

C'est un tableau de 6 pointeurs vers des chaines de caractères.

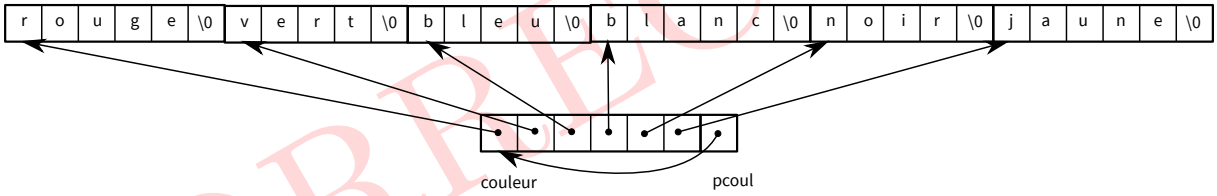
Du moins, c'est l'explication classique, mais en fait pas tout à fait. J'ai dit en amphi que la différence entre `char*` et `char[]` est l'une des choses les plus obscures du langage C, l'une des seules choses qui file des remords aux concepteurs.

La vérité, c'est que le standard dit que quand on utilise un tableau dans une expression, il se fait passer pour un pointeur vers la première case du tableau (sauf en argument de `sizeof()`, de `&` ou quand c'est une chaine de caractères pour initialiser un pointeur). Cette page l'explique très bien : `http://openclassrooms.com/courses/la-verite-sur-les-tableaux-et-pointeurs-en-c`

Donc, couleur est bien un tableau de pointeurs comme sur le schéma ci-dessous, mais quand on utilise couleur dans une expression, il se fait passer pour l'adresse de sa première case. C'est assez différent de `pcoul`, qui lui est un pointeur, un vrai. C'est donc un bout de mémoire où est écrit une adresse.



Si on se sent l'humeur taquine, on peut faire remarquer qu'il est probable que les différentes chaines de caractères soient consécutives en mémoire. Cela donne le schéma suivant :



Fin réponse

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

Réponse

C'est l'adresse de la seconde case du tableau, c'est-à-dire `&(couleur[2])`

Fin réponse

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

Réponse

C'est `couleur[0]`, c'est à dire l'adresse de la chaîne `"rouge"`, qui se trouve quelque part dans le segment data du processus (là où il y a le code et les globales).

Fin réponse

▷ **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, on décale de deux cases, et ça pointera sur 'u'. La fin de la chaine est après 'e' de "rouge".

Fin réponse

▷ **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) = "jaune" , "jaune" étant situé à couleur[5]
  *(couleur + 5) +3 = "ne"

*(*(couleur + 5) +4) = 'e'
*(*(couleur + 5) +5) = '\0'
*(*(couleur + 5) +6) = 'c' , c quelconque
```

Fin réponse

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

Soit le type suivant :

```
typedef struct {
  char nom[MAX_CAR];
  int  rayon;
} planete_t;
```

▷ **Question 1:** Écrire un constructeur pour ce type `planete_t`.

Réponse

```
planete_t *planete_create(char *nom, int rayon) {
    planete\_t *p1;
    p1 = (planete_t *) malloc (sizeof(planete_t));
    strcpy(p1->nom, nom);
    p1->rayon = rayon;
    return p1;
}
```

Fin réponse

▷ **Question 2:** Écrire une fonction qui saisit les données relatives à une planète, et crée une planète en conséquence.

Réponse

```
planete_t *planete_saisir(){
  char nom[32];
  int rayon;

  printf("Donner nom planète : ");
  scanf ("%s",nom);
  printf("Donner son rayon : ");
  scanf ("%d",&rayon);
  return planete_create(nom,rayon);
}
```

Fin réponse

▷ **Question 3:** Écrire une fonction qui duplique une planète (un *copy constructor*).

Réponse

```
// Version compliquée

planete_t* planete_dup(planete_t *p){

  planete_t *res;
  res=(planete_t*)
     malloc(sizeof(planete_t));

  strcpy(res->nom, p->nom);
  p1->rayon = p->rayon;

  return p1;
}
```

```
// Version simple

planete_t* planete_dup(planete_t *p){

  return planete_create(p->nom,
                        p->rayon);
}
```

Fin réponse

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

```
void main() {

  planete_t p;
  planete_t *ptr_p;
  ptr_p = planete_saisir();
  p = *ptr_p;
  printf("%s %d \n", p.nom, p.rayon);
  ptr_p = planete_dup(p);
  printf("%s %d \n", ptr_p->nom, ptr_p->rayon);
}
```

Comment s'analysent les types des différentes références suivantes :

Référence	Type
ptr_p	
*ptr_p	
ptr_p->nom	
ptr_p->rayon	
p	
&p	
p.nom	
p.rayon	

Réponse

Référence	Type
ptr_p	planete_t *
*ptr_p	planete_t
ptr_p->nom	char []
ptr_p->rayon	int
p	planete_t
&p	planete_t*
p.nom	char []
p.rayon	int

Fin réponse

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

▷ **Question 1:** 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)
```

▷ **Question 2:** 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);
}
```

Fin réponse

★ **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).

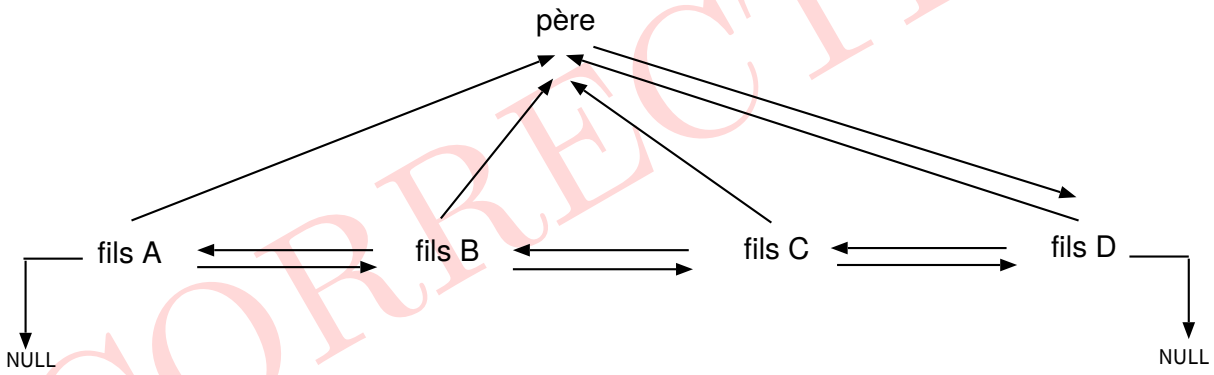


FIGURE 1 – *Filiation sous Unix*

▷ **Question 1:** Représenter schématiquement, 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;
  }
}
```

Fin réponse