Travaux Dirigés de C n°6

Cours de langage et programmation

—L3 Informatique—

Déclarations de types compexes

Les exercices de cette séance permettront de voir quelques applications plus complexesde la notion de pointeur. Nous verrons en particulier les notions de pointeur sur fonction et de type désincarné, l'allocation de mémoire et la création de type.

▸ **Exercice 1.** *(Pointeur et fonction 1) Déclarer une fonction f sans argument retournant l'adresse d'une procédure sans argument. On rappelle qu'une procédure est une fonction ne renvoyant aucune valeur utile, c'est-à-dire renvoyant un objet de type void.*

▸ **Exercice 2.** *En utilisant l'opérateur* `typedef`*, déclarer en C les types suivants :*

1. *Pointeur sur pointeur sur* `char`*,*
2. *Pointeur sur tableau de* `char`*,*
3. *Pointeur sur un tableau de 7* `char`*,*
4. *Pointeur sur une fonction ne prenant pas de paramètre et retournant* `void`*,*
5. *Tableau de 12 pointeurs sur fonction retournant un pointeur sur un tableau de 7* `char`*, et prenant en parametre un* `int`*.*

▸ **Exercice 3.** *Déclarez une fonction*

1. *qui prend en argument un pointeur sur un tableau de 3 entiers et qui retourne un pointeur sur un tableau de 3 entiers,*
2. *qui prend en argument un tableau de 3 entiers et qui retourne un tableau de 3 entiers,*
3. *qui prend en argument un pointeur sur une fonction qui prend en argument un entier et retourne un pointeur sur une fonction qui retourne un* `char` *et qui prend en argument un* `short`*, et qui retourne son argument.*

▸ **Exercice 4.** *(type désincarné pour l'allocation dynamique) On veut réserver la place mémoire pour N objets de type "tableau de 5 pointeurs sur entiers". Écrire l'allocation d'un tel bloc. Faire la même déclaration pour un tableau statique.*

▸ **Exercice 5.** *(Fonction et pointeur 2 : types désincarnés pour le prototypage) Supposons que, dans le cadre d'un programme de type calcul formel, on veuille déclarer une fonction* Δ *qui renvoie la dérivée par rapport à* x *ou* y *d'une fonction à deux variables réelles* $f(x, y)$ *à valeurs dans* $\mathbb{R}$*.*

1. *Déclarer un identificateur* `f` *de type "pointeur sur fonction à deux arguments réels retournant un réel".*
2. *En déduire le type désincarné correspondant à f.*
3. *Déclarer l'identificateur* `Delta`*. Le choix de la variable de dérivation (x ou y) pourra se faire par exemple en passant un argument entier à la fonction* `Delta` *: si* `argnum`*=1, on dérive par rapport à la première variable, si* `argnum`*=2, on dérive par rapport à la deuxième...).*
4. *Déclarer le type désincarné permettant de décrire un pointeur vers un objet de même type que Delta.*

▸ **Exercice 6.** *(Création de type) Pour simplifier l'écriture dans l'exercice précédent, utiliser le déclarateur* `typedef` *pour :*

1. *Créer un type* `FONCTION` *représentant une "fonction prenant deux arguments réels et retournant un réel".*
2. *Créer un type* `OPERATEUR` *représentant une "fonction prenant pour argument un entier et un pointeur sur* `FONCTION` *et renvoyant un pointeur sur* `FONCTION`*".*
3. *Réécrire le type demandé à la dernière question de l'exercice précédent.*
4. *On définit :*

```
double x,y; FONCTION g; OPERATEUR Delta;

double g(double x, double y) {
  return <{\rm \it valeur de $g(x,y)$}>; }

FONCTION *Delta(int argnum, FONCTION *f) {
  if (argnum==1) return (\& {\rm ($\frac{\partial (*f)}{\partial x}$)});
  if (argnum==2) return (\& {\rm ($\frac{\partial (*f)}{\partial y}$)}); }
```

C'est-à-dire, `g` *est une fonction à deux variables réelles et* `Delta` *est l'opérateur de dérivée partielle. Que vaut alors :* `*Delta(1,Delta(2,&g))(x,y)` *?*

▸ **Exercice 7.** *(Fonction s'appelant elle-même) Que fait le programme suivant ?*

```
#include <stdio.h>

int f(int n, int (*f)()) {
  return n>0?n*f(n-1,f):1;
}

int main() {
  int n;
  printf("donner n :"); scanf("%d",&n);
  printf("f(%d)=%d\n",n,f(n,f));
}
```

▸ **Exercice 8.** *(mapping) Écrire une fonction* `map` *qui applique une fonction sur un tableau d'entiers. La fonction* `map` *prend en paramètre une fonction* `f`*, un tableau d'entiers ainsi que la taille du tableau. La fonction* `f` *prend un entier en paramètre et retourne un entier. Si le tableau contient les valeurs x_0, x_1, x_2, ..., x_n, le tableau contient après l'appel les valeurs* `f`(x_0), `f`(x_1), `f`(x_2), ..., `f`(x_n)*. Tester avec la fonction $f(x) = x^2$.*

▸ **Exercice 9.** *(itération de fonction) Écrire une fonction* `itarray` *qui calcule l'itération d'une fonction à deux arguments sur un tableau d'entiers. La fonction* `itarray` *prend en paramètre une fonction* `f`*, un tableau d'entiers ainsi que la taille du tableau. La fonction* `f` *prend deux entiers en paramètre et retourne un entier. Si le tableau contient les valeurs x_0, x_1, x_2, ..., x_n, la fonction* `itarray` *retourne la valeur* `f` *(*`f` *(...*`f` *(*`f` *(x_0, x_1), x_3),...)x_0). Tester avec les fonctions $f(x, y) = \max(x, y)$ et $f(x, y) = x+y$. Que renvoie la fonction* `itarray` *dans chacun des deux cas ?*