0.1 Pointeurs et Entrées/Sorties

0.1.1 1. Les Pointeurs :

0.1.2 1.1. Un peu de théorie :

La mémoire peut-être vue comme un ensemble de cases, nommées octets. Chaque octet comprend 8 bits qui peuvent chacun avoir la valeur 0 ou 1. Cet ensemble se présente comme un tableau d'octets, dont les indices sont des valeurs entières : les adresses. Toute valeur manipulée dans un programme est représentable en mémoire et occupe un certain nombre d'octets, dépendant de son type. Ainsi un entier peut être représenté sur 1, 2, 4 ou même 8 octets ; un caractère est représenté sur 1 octet ; un réel est représenté sur 4 ou 8 octets... Lorsque dans un programme C vous déclarez : **int a**, vous réservez en fait un espace contiguë de 2 octets (16 bits). Cet espace a une adresse, comme indiquée précédemment, vous pouvez même en connaître la valeur grâce à : **&a**. Plus généralement, il est possible d'appliquer l'opérateur **&** à toute expression qui pourrait constituer la partie gauche d'une affectation. Un pointeur est une variable qui peut contenir l'adresse d'une autre variable ou plus généralement sa référence. Un pointeur permet d'accéder indirectement à un objet, on connaît l'adresse donc on peut connaître l'objet. Un pointeur est typé, en effet, le **&a** ne renvoie que l'adresse du premier octet de **a** (qui est codé sur 2 octets contigus), il faut donc connaître le type (donc la taille) de **a** pour savoir combien d'octets récupérer en plus, après **&a**.

0.1.3 1.2. Le type pointeur :

- **Le type d'un pointeur :** Le type d'un pointeur est donné explicitement par l'objet pointé. Si **a** est un entier et **pa** son pointeur lors, **pa** est de type pointeur d'entier.
- **Opérateur &** C'est un opérateur unaire, il s'applique à tout objet pouvant être en partie gauche d'une affectation. Il renvoie l'adresse du premier octet de l'objet, attention **&(&x)** n'a pas de sens.
- **Opérateur *** C'est l'opérateur d'indirection, s'applique sur un pointeur et renvoie l'objet situé à l'adresse indiquée par le pointeur (en tenant compte de sa taille). Notez que ***(&a)** et **a** désigne le même objet.
- **Déclaration de pointeur :** Pour déclarer un entier vous faites **int a** ? ce qui signifie : a est un entier codé sur 2 octets. Avec les pointeurs c'est équivalent, pour déclarer un pointeur vous écrirez **int *pa**, ce qui signifiera ***pa** est un entier codé sur 2 octets donc **pa** est un pointeur d'entier sur 2 octets. Notez que dans un cas il y'a un ***** et dans l'autre non. Une autre façon de voir les choses est de dire qu'a int le type entier, on associe **(int *)** le type pointeur d'entier. Comme pour le déclaration habituelle, il n'y a que réservation de la mémoire et non-initialisation. Il est bien-sûr possible d'avoir des pointeurs de pointeurs :

  ```
  int x,*px ;
  int * *ppx ;
  ```

- **Manipulations des pointeurs :** Le mieux est encore de prendre un exemple

  ```
  int x,y,*px ;
  x=0 ;
  y=1 ;
  ```

```
p=&x ;
y=*p ;
*p=*p+10 ;
```

A votre avis que vaut x, y ? et sur quoi pointe p ? pour le savoir exécutez le programme suivant :

```
#include <stdio.h>
int main (void)
{
int x,y,*p ;
x=0 ;
y=1 ;
p=&x ;
y=*p ;
*p=*p+10 ;
printf("\n la variable x vaut %d", x);
printf("\n la variable y vaut %d", y);
if (*p== x ) { printf("\n le pointeur p contient l'adresse
   de x\n");}
else { printf("\n le pointeur p contient l'adresse
   de y\n");}
return(1);
}
```

- **Opérations sur les pointeurs :**
  - **NULL** est une valeur constante de pointeur qui signifie que le pointeur ne pointe vers rien. Il est possible de faire p=NULL. A la déclaration, tous les pointeurs contiennent NULL.
  - Il est possible affecter la valeur d'un pointeur à un autre pointeur du même type :

    ```
    int x,*q,*p ;
    p=&x ;
    q=p ;
    ```

  - Mais cela est impossible entre pointeurs de types différents, pour cela il faut convertir explicitement l'un des types :

    ```
    long x,*px ;
    char *pc ;
    px=&x ;
    pc = (char *)px ;
    ```

  - De même, il est impossible d'affecter une valeur entière à un pointeur, car un pointeur est de type pointeur de quelques choses. Néanmoins, en utilisant une conversion explicite on y parvient :

    ```
    long *px ;
    px = (long *) 0x43 ; #en notation hexadecimale
    ```

– Il est possible en revanche d'incrémenter la valeur d'un pointeur, mais attention rajouter 1 à un pointeur d'entier sur 4 octets, augmente sa valeur de 4 octets. Idem pour la soustraction.
– Il est également possible de comparer les valeurs de pointeurs :

```
point1 ==(int *) 0x80 ;
*point2 == 15 ;
point3 <= point4 ;
```

– S'il n'est pas possible de déclarer des variables de type void , il est en revanche possible de déclarer des variables de type pointeur de void. Ces pointeurs ne sont pas déréférençables, mais sont compatibles (sans besoin de conversion explicite) avec tous les types pointeurs.

0.1.4 1.3. Pointeur et tableau :

Quand on déclare `long Tab [15];` , on indique que Tab est un tableau de 15 entiers longs. On écrit Tab[i] pour accéder à l'élément de rang i. Tab est aussi une constante, dont la valeur est une adresse (l'adresse de début du tableau, c'est-à-dire celle de son premier élément, d'indice 0) et dont le type est pointeur de long. De même d'après l'arithmétique sur les pointeurs et le rangement contigu des éléments d'un tableau, on a donc la série d'égalités suivante :

```
&(Tab[ i ]) équivaut à Tab + i
Tab[ i ] équivaut à *(Tab+i)
&(Tab[0]) équivaut à Tab
```

Néanmoins, un pointeur est une variable et le nom du premier élément d'un tableau est une constante. Ceci implique que, si ptab est un pointeur de long : `Tab++` et `Tab=ptab` sont interdits mais `ptab = Tab` est autorisé.

0.1.5 1.4. Pointeur de structure :

En C il est possible d'utiliser des structures, comme leur nom l'indique, il s'agit d'un ensemble formé d'ensembles plus simples. Par exemple :

```
struct {
int i ;
double f ;
} Doublet ;
```

Défini Doublet, une structure à 2 champs, le premier champ est un entier et le second un double. Pour accéder à un champ de la structure, vous devez utiliser le sélecteur "." :

```
Doublet.i = 1 ;
Doublet.f = 1.2 ;
```

Il est possible de définir des pointeurs de structure, par exemple :

```
struct Doublet *pDoublet ;
```

A partir du pointeur, il est possible d'accéder à l'un des champs de la structure pointée par le sélecteur "->" (flèche) :

```
pDoublet -> i = 1 ;
pDoublet -> f = 1.2 ;
```

Notez que j'aurais très bien pu écrire :

```
(*pDoublet).i = 1 ;
(*pDoublet).f = 1.2 ;
```

0.1.6 1.5. Allocation dynamique de la mémoire :

Il existe 3 fonctions particulières qui permettent de manipuler explicitement la mémoire, pour les utiliser vous devez inclure le fichier stdlib.h :

- **malloc(taille) :** retourne un pointeur sur un espace mémoire reservé de taille : "taille" ou bien NULL si la demande ne peut être satisfaite.
- **calloc(nbre_objet,taille) :** retourne un pointeur sur un espace mémoire réservé à un tableau de "nbre_objet" de taille "taille". Ou bien NULL si la demande ne peut être satisfaite.
- **free(pointeur) :** libère l'espace pointé par p mais ne fait rien si p vaut NULL.

Voilà un exemple d'utilisation à compiler :

```
#include <stdio.h>
#include <stdlib.h>
int main (void)
{
int *pointeur;
printf("\n Allocation dynamique de la mémoire ...");
pointeur=(int *) malloc (sizeof(int));
printf("\n On teste l'allocation ...");
if (pointeur==NULL) {printf("\n Pb d'allocation!"); exit(1);}
printf("\n Allocation reussie!");
*pointeur= 10;
printf("\n Libération de la mémoire ... \n");
free(pointeur);
return(1);
}
```

Il devrait vous afficher ceci :

```
Allocation dynamique de la mémoire ...
On teste l'allocation ...
Allocation reussie!
Libération de la mémoire ...
```

0.1.7 2. Les Entrées/Sorties :

Le but de cette partie est d'indiquer comment, sont gérées les entrées/sorties dans un fichier en C. Toutes manipulations sur les fichiers nécessitent en C l'inclusion de la bibliothèque stdio.h et pour les chaînes de caractères c'est string.h. Je débute par un exemple pour fixer les idées :

```
#include <stdio.h>
#include <string.h>
int main (void)
{
struct {
char nom[20];
char prenom[20];
} fiche;
FILE *fichier;/* fichier est le nom local du fichier */
fichier=fopen("data.txt","r");/* initialisation de fichier,
   qui représentera, le
fichier data.txt ouvert en lecture */
/*Vérification d'usage */
if(fichier==NULL){printf("Pb d'ouverture de fichier");
   exit(1);}
printf("Liste des noms et prénoms ...");
while (!feof(fichier))/* tant qu'on n'est pas à la fin
   du fichier */
{
/* on récupère le nom et le prénom de l'individu
et les stockent dans une structure */
fscanf(fichier,"%s %s", &fiche.nom, &fiche.prenom);
/* Aussitôt on l'affiche à l'écran */
printf("%s %s \n", fiche.nom, fiche.prenom);
fflush(stdout);/* forcer l'écriture à l'écran */
}
fclose(fichier);/* on ferme le fichier */
return(1);
}
```

- **fichier :** Il s'agit d'un pointeur de fichier c'est à partir de lui que je manipule le fichier data.txt.
- **fopen :** C'est la fonction qui me permet d'ouvrir un fichier, vous indiquez d'abord le chemin relatif vers le fichier, puis le mode d'ouverture. w : création ou réinitialisation, r : lecture, a : création ou ajout en fin de fichier. Comme il s'agit d'une allocation dynamique, il convient de vérifier qu'elle s'est bien terminée.
- **feof(fichier)** Renvoie 0 si on est en fin de fichier et un autre entier après la première tentative de lecture au-delà de la fin.
- **fscanf** fscanf(fichier,"format",&arg) et fprintf(fichier,"format",arg) sont les équivalents de fscanf("format",&arg) et fprintf("format",arg) mais pour les fichiers. Leur utilisation est exactement la même. Le champ fichier permet de spécifier quel fichier est manipulé.
- **fflush** fflush(stdout) force l'écriture à l'écran, pour fichier j'aurais mis fflush(fichier).
- **fclose(fichier)** Ferme fichier.

Le fichier data.txt est le suivant :

François Dupont
Claude Vrille
Michel Tomate
Laurent Tambour

L'exécution du programme affiche :

Liste des noms et prénoms ...
François Dupont
Claude Vrille
Michel Tomate
Laurent Tambour
Laurent Tambour

«« Précédent[1] Suivant »»[2]

[1] http://www.trustonme.net/didactels/154.html
[2] http://www.trustonme.net/didactels/156.html