Chapitre III : Les Pointeurs

1. Introduction

La mémoire centrale utilisée par les programmes, est découpée en octets. Chacun de ces octets est identifié par un numéro séquentiel appelé adresse. Par convention, une adresse est notée en hexadécimal et précédée par**0x.**

0x3fffd10	
0x3fffd11	
0x3fffd12	
0x3fffd13	
0x3fffd14	
0x3fffd15	
0x3fffd16	
0x3fffd17	

Déclarer une variable, c'est attribuer un nom (l'identificateur) à une zone de la mémoire centrale. Cette zone est définie par :

- sa position c'est-à-dire l'adresse de son premier o ctet
- sa taille c'est-à-dire le nombre d'octets

short int toto = 18;

0x3fffd10		
0x3fffd11		
0x3fffd12		
0x3fffd13		
0x3fffd14	18	toto
0x3fffd15		
0x3fffd16		
0x3fffd17		

…

Pour accéder à la valeur contenue dans une variable, on utilise tout simplement son nom.

Mais il peut arriver qu'on veuille accéder à l'adresse d'une variable. Dans ce cas, on utilise **l'opérateur d'adresse &**(notation C) suivi du nom de la variable. **&toto** vaut **0x3fffd14** (adresse du premier octet de toto)

Inconvénients des variables statiques

Les variables « classiques », déclarées avant l'exécution d'un programme, pour ranger les valeurs nécessaires à ce programme, sont des variables statiques, c'est à dire que la place qui leur est réservée en mémoire est figée durant toutel'exécution du programme. Ceci a deux conséquences :

- Risque de manquer de place si la place réservée (par exemple le nombre d'éléments d'un tableau) est trop petite. Il faut alors que le programme prenne en charge le contrôle du débordement.
- Risque de risque de gaspiller de la place si la place réservée est beaucoup plus grande que celle qui est effectivement utilisée par le programme.

Un cahier, dont le nombre de pages est fixé a priori, fournit une bonne image d'une variable statique.

2. Les pointeurs

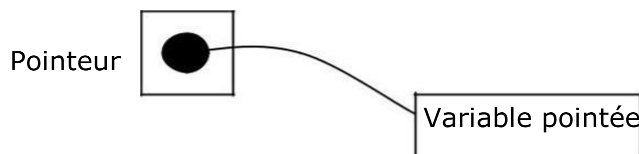
Un pointeur ou variable dynamique est une variable qui contient l'adresse d'une autre variable. Il permet de désigner directement une zone de la mémoire et donc l'objet dont la valeur est rangée à cet endroit.Un pointeur est souvent typé de manière à préciser quel type d'objet il désigne dans la mémoire.

2.1. Pourquoi les pointeurs

- La mémoire d'un ordinateur est limitée en taille.
- Ne réserver de la mémoire qu'en cas de nécessité.
- Libérer cette mémoire une fois le traitement terminé.
- Cette place libérée est ainsi disponible pour une utreutilisation dans le même programme.

Un classeur dans lequel on peut ajouter ou retirer des pages est une bonne image d'une variable dynamique.
La notion de pointeurs est très commode dans les langages de programmation, car elle permet de représenter simplement des structures de données dynamiques comme les piles, les files, les listes, les arbres etc.

L'adresse contenue dans un pointeur est celle d'une variable qu'on appelle variable pointée.
On dit que le pointeur pointe sur la variable dont il contient l'adresse.



Un pointeur est associé à un type de variable sur lequel il peut pointer. Par exemple, un pointeur sur entier ne peut pointer que sur des variables entières.

- Un pointeur est lui-même une variable et à ce titreil possède une adresse.
- Il convient de ne pas confondre l'adresse de la variable pointeur et l'adressecontenue dans le pointeur (adresse de la variable pointée)

2.2. Déclaration d'un pointeur

- Par convention, les pointeurs utilise le symbole ^ avant la variable pointée.
- Il faut déclarer le type de la variable pointée.

en algorithmique	en langage C
nom : ^ *type pointé*	*type* nom;*
exemple :	*exemple :*
ptoto: ^ entier	int * ptoto;
	* est appelé opérateur de déréférencement

2.3. Utilisation d'un pointeur

On utilise un pointeur pour mémoriser l'emplacement d'une autre variable.

- Il est très rare d'affecter directement une adresse à un pointeur. On affecte en
- général l'adresse d'une variable existante.
- Pour cela, en algorithmique, on utilise l'expression "adresse de" alors qu'en C, on utilise l'opérateur d'adresse &

Exemple:

-

Algorithmique	Langage C
Variables	
va : entier	int va;
pent: ^ entier	int * pent;
pcar: ^caractère	char * pcar;
Début	
pcar ← 0x3fffd14 //affectation directe	pcar = 0x3fffd14; //affectation directe
pent ← **adresse** de va	pent = &va;
…	…

Il est possible d'extraire la valeur d'une variablepointée.

2.4.Déréférencer un pointeur consiste à extraireal valeur de la variable sur laquelle il pointe.

- On peut aussi modifier la valeur d'une variable pointée à travers un pointeur déréférencé (accès en écriture)

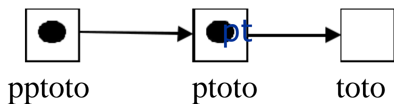
Algorithmique	

Variables
n: entier
p: ^entier
Début
n ← 33
p ← adresse de n *// p pointe sur n*
Ecrire (*p)*//la valeur 33 est Affichée*
*p ← 34 *// n vaut maintenant 34*
Ecrire (n)*//la Valeur 34 est Affichée*
Fin
	#include <stdio.h>
	#include <string.h>
	int main ()
	{
	int n = 33;
	int *p; //déclaration du pointeur p
	p = &n;
	printf ("%d \n",*p); //affiche 33 à l'écran, la valeur de n
	*p = 34;
	printf ("%d \n",n);
	}

- Attention à toujours initialiser un pointeur. Un po inteur qui n'est pas initialisé s'appelle un **pointeur pendant**. Un pointeur pendant ne pointe pas nulle part mais n'importe où. Si l'on déréférence ce pointeur et qu'on affecte eunnouvelle valeur, on va écraser un emplacement mémoire quelconque, et on risque de faire planter le programme.
- Si on veut que le pointeur ne pointe nulle part, il faut l'initialiser à NIL (Not Identified Link) ou NULL en C. C'est l'équivalent de 0 pour les pointeurs. Mais attention, il ne faut jamais déréférencer un pointeur nul. Avant dedéréférencer un pointeur, il faut toujours s'assurer qu'il n'est pas nul.

p: pointeur sur entier p NIL //p ne pointe nulle part	int *p = NULL;

Le fait qu'un pointeur pointe sur une variable s'appelle **indirection** (comme accès **indirect**). Quand un pointeur pointe sur un autre pointeur, il y a **double indirection**.



On peut accéder à toto par pptoto en utilisant deux fois l'opérateur de déréférencement *

* *pptoto est équivalent à toto et à *ptoto

On pourrait imaginer de la même façon des indirections triples voire quadruples.

3. Gestion dynamique de la mémoire

3.1. Allocation / désallocation d'une zone de mémoire

On a défini un type t.
On déclare : ptr : pointeur_sur_t ou ^t
Création d'une variable dynamique de type t allouer (ptr) réserve un emplacement mémoire de la taille correspondant au type t, met dans la variable ptr l'adresse de la zone mémoire qui a été réservée.
L'emplacement pointé par ptr sera accessible par ptr ^.
ptr : ^t
En C :

```
#include <stdlib.h> int *p;
p=((int*) malloc sizeof(int)); *p=1;
```

Libération de la place occupée par une variable dynamique
desallouer (ptr) libère la place de la zone mémoire dont l'adresse est dans ptr (et la rend disponible pour l'allocation d'autres variables) laisse la valeur du pointeur en l'état (n'efface pas l'adresse qui est dans la variable pointeur).
ptr^ : t

desallouer (ptr)
Remarque
Si on fait appel au pointeur désalloué, il renvoie une information qui n'a aucun sens.

En c

Ceci se fait à l'aide de l'instruction **free** qui a pour syntaxe

free (nom-du-pointeur)
L'instruction free (p) libère la mémoire utilisée arp l'objet pointé par p
A toute instruction de type malloc doit être associée une instruction de type free
Attention, si la mémoire a été allouée par une déclaration, c'est le compilateur qui se charge de libérer la mémoire lorsque l'on sort de la fonction.
Il est aussi important d'allouer correctement la mémoire que de libérer correctement la mémoire.

3.2. Arithmétique des pointeurs

Comme les pointeurs jouent un rôle si important, le langage C soutient une série d'opérations arithmétiques sur les pointeurs que l'on ne rencontre en général que dans les langages machines. Le confort de ces opérations en C est basé sur le principe suivant:

Toutes les opérations avec les pointeurs tiennent compte automatiquement du type et de la grandeur desobjets pointés.

3.2.1. Affectation par un pointeur sur le même type

Soient P1 et P2 deux pointeurs sur le même type de données, alors l'instruction **P1 = P2**; fait pointer P1 sur le même objet que P2

3.2.2.Addition et soustraction d'un nombre entier

Si P pointe sur l'élément A[i] d'un tableau, alors
P+n pointe sur A[i+n]
p-n pointe sur A[i-n]

3.2.3. Incrémentation et décrémentation d'un pointeur

Si P pointe sur l'élément A[i] d'un tableau, alors après l'instruction
P++ ; P pointe sur A[i+1]
P+ =n ; P pointe sur A[i+n]
P-- ; P pointe sur A[i-1]
P-=n ; P pointe sur A[i-n]

- L'addition, la soustraction, l'incrémentation et la décrémentation sur les pointeurs sont seulement définies *à l'intérieur d'un tableau*. Si l'adresse formée par le pointeur et l'indice sort du domaine du tableau, alors le résultat n'est pas défini.

3.2.4. Soustraction de deux pointeurs

Soient P1 et P2 deux pointeurs qui pointent dans le même tableau:
P1-P2 fournit le nombre de composantes comprises entre P1 et P2

Le résultat de la soustraction P1-P2 est
- négatif, si P1 précède P2
- zéro, si P1 = P2
- positif, si P2 précède P1
- indéfini, si P1 et P2 ne pointent pas dans le même tableau

Plus généralement, la soustraction de deux pointeurs qui pointent dans le même tableau est équivalente à la soustraction des indices correspondants.

3.2.5. Comparaison de deux pointeurs

On peut comparer deux pointeurs par **<, >, <=, >=, ==, !=**.

La comparaison de deux pointeurs qui pointent *dans le même tableau*est équivalente à la comparaison des indices correspondants. (Si les pointeurs ne pointent pas dans le même tableau, alors le résultat est donné par leurs posit ons relatives dans la mémoire).

3.3. Pointeurs et tableaux à une dimension

Tout tableau en C est en fait un pointeur constant. Dans la déclaration
int tab[10];
tab est un pointeur constant (non modifiable) dont la valeur est l'adresse du premier élément du tableau. Autrement dit, tab a pour valeur &tab[0]. On peut donc utiliser un pointeur initialisé à tab pour parcourir les éléments du tableau.

Exemple :

```
#include <stdio.h>
#define N 5
int tab[5] = {0, 1, 4, 20, 7};
main ()
{
int i;
int *p;
p = tab;
for (i = 0; i < N; i++)
{
printf(" %d \n",*p);
p++;
}
}
```

On accède à l'élément d'indice i du tableau tab grâce à l'opérateur d'indexation [], par l'expression tab[i]. Cet opérateur d'indexation peut en fait s'appliquer à tout objet p de type pointeur. Il est lié à l'opérateur d'indirection *par la formule
p[i] = *(p + i)
Les pointeurs et tableaux se manipulent donc exactement de la même manière. Par exemple,

le programme précédent peut aussi s'écrire

```
#include <stdio.h>
#define N 5
int tab[5] = {0, 1, 4, 20, 7};
main()
{
int i;
int *p;
p = tab;
for (i = 0; i < N; i++)
printf(" %d \n", p[i]);
}
```

Cependant, la manipulation de tableaux, et non de pointeurs, possède certains inconvénients dus au fait qu'un tableau est un pointeur constant. Ainsi

- on ne peut pas créer de tableaux dont la taille estune variable du programme,
- on ne peut pas créer de tableaux bidimensionnels dont les lignes n'ont pas toutes le même nombre d'éléments.

Ces opérations deviennent possibles dès que l'on manipule des pointeurs alloués dynamiquement. Ainsi, pour créer un tableau d'entiers à n éléments où n est une variable du programme, on écrit :

```
#include <stdlib.h>
main()
{
int n;
int *tab;
...
tab = (int*)malloc(n * sizeof(int));
...
free(tab);
}
```

Les éléments de tab sont manipulés avec l'opérateurd'indexation [], exactement comme pour les tableaux.
Les deux différences principales entre un tableau et un pointeur sont

- un pointeur doit toujours être initialisé, soit parune allocation dynamique, soit par affectation d'une expression adresse, par exemple p = &i ;
- un tableau n'est pas une Lvalue ; il ne peut donc pas figurer à gauche d'un opérateur d'affectation. En particulier, un tableau ne supporte pas l'arithmétique (on ne peut pas écrire tab++;).

3.4. Pointeurs et tableaux à plusieurs dimensions

Un tableau à deux dimensions est, par définition, un tableau de tableaux. Il s'agit donc en fait d'un pointeur vers un pointeur. Considérons le tableau à deux dimensions défini par :

int tab[M][N];
tab est un pointeur, qui pointe vers un objet lui-même de type pointeur d'entier.
tab a une valeur constante égale à l'adresse du premier élément du tableau, &tab[0][0].
De même tab[i], pour i entre 0 et M-1, est un pointeurconstant vers un objet de type entier, qui est le premier élément de la ligne d'indice i. l'**é**lment tab[i] a donc une valeur constante qui est égale à &tab[i][0].
Exactement comme pour les tableaux à une dimension, les pointeurs de pointeurs ont de nombreux avantages sur les tableaux multi-dimensionnés. On déclare un pointeur qui pointe sur un objet de type *type* * (deux dimensions) de la même manière qu'un pointeur, c'est-à-dire
***type **nom-du-pointeur*;**
De même, un pointeur qui pointe sur un objet de type *type* ** (équivalent à un tableau à 3 dimensions) se déclare par :
***type ***nom-du-pointeur*;**
Par exemple, pour créer avec un pointeur de pointeur une matrice à k lignes et n colonnes à coefficients entiers, on écrit :

```
main()
{
int k, n;
int **tab;
tab = (int**) malloc(k * sizeof(int*));
for (i = 0; i < k; i++)
tab[i] = (int*) malloc (n * sizeof(int));
....
for (i = 0; i < k; i++)
free(tab[i]);
free(tab);
}
```

La première allocation dynamique réserve pour l'objet pointé par tab l'espace mémoire correspondant à k pointeurs sur des entiers. Ces k pointeurs correspondent aux lignes de la matrice. Les allocations dynamiques suivantes réservent pour chaque pointeur tab[i] l'espace mémoire nécessaire pour stocker n entiers.
Contrairement aux tableaux à deux dimensions, on pe ut choisir des tailles différentes pour chacune des lignes tab[i]. Par exemple, si l'on veut que tab[i] contienne exactement i+1éléments, on écrit

```
for (i = 0; i < k; i++)
tab[i] = (int*) malloc ((i + 1) * sizeof(int));
```

3.5. Pointeurs et chaînes de caractères

On a vu précédemment qu'une chaîne de caractères aitét un tableau à une dimension d'objets de type char, se terminant par le caractère nul '\0. On peut donc manipuler toute chaîne de caractères à l'aide d'un pointeur sur un objet de type char. On peut faire subir à une chaîne définie par char *chaine; des affectations comme chaine = "ceci est une chaine"; et toute opération valide sur les pointeurs, comme l'instruction chaine++;.
Ainsi, le programme suivant imprime le nombre de caractères d'une chaîne (sans compter le caractère nul).

```
#include <stdio.h>
main()
{
int i;
```