Les pointeurs

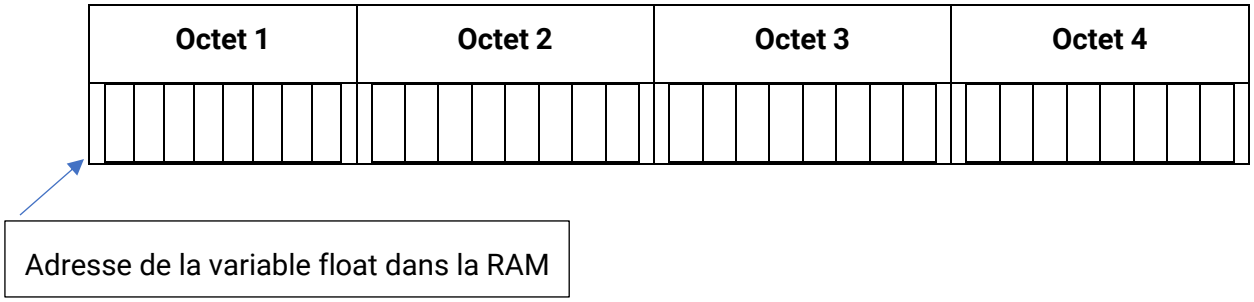
La RAM et les adresse

La RAM ou la mémoire centrale d'un ordinateur est composée d'un très grand nombre d'octets. Chaque octet est repéré par un numéro appelé adresse de l'octet.

Chaque variable dans la mémoire occupe des octets contigus, c'est-à-dire des octets qui se suivent.

Par exemple, un float occupe 4 octets qui se suivent. L'adresse de la variable est l'adresse de son premier octet.

Une variable flaot dans la RAM



Chaque variable dans la RAM est accessible à partir de son adresse physique. C'est-à-dire, le système d'exploitation accède à une variable dans la RAM à partir de son adresse physique.

Même chose pour les fichiers sur le disque dur, le système d'exploitation accède à un fichier sur le disque dur à partir de son adresse.

Voici un schéma très simplifié expliquant ce qui se passe au niveau de la RAM

Tableau 1

Nom variable	Adresse dans la RAM
X	0x100000e29
Y	0x100000e30
Z	0x100000e28

RAM

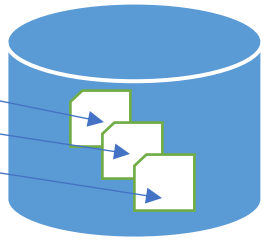
126
789
56

Par exemple, depuis le tableau 1, le système d'exploitation utilise le nom de la variable X, puis identifie son adresse pour accéder à son emplacement dans la RAM

Même chose pour l'accès au disque dur

Tableau 2

Nom fichier	Adresse dans la RAM
cv.doc	0x100000e29
Cours.doc	0x100000e30
Exemple.c	0x100000e28



Depuis le tableau 2, le système d'exploitation utilise l'adresse physique du fichier cv.doc pour accéder à cet emplacement sur le disque dur.

En C, vous avez constaté que dans l'instruction scanf , pour lire la valeur de la variable X de type entier, nous utilisons la notation suivante :

Soit les instructions suivantes :

1) int x;
2) scanf("%d",&x);

Dans l'instruction 1, nous déclarons une variable nommée X de type entier.

Dans l'instruction 2, nous appelons la fonctions scanf pour lire un entier.

La fonction scanf prend la valeur tapée depuis le clavier et la range dans une case dans la RAM ayant l'adresse de X.

Qu'est-ce qu'un pointeur ?

L'adresse d'une variable peut elle-même être mémorisée dans une variable. Les variables dont les valeurs sont des adresses s'appellent des pointeurs

Soit l'exemple suivant :

On souhaite déclarer une variable qui s'appelle X ayant la valeur 20 :

```
int X = 20 ;
```

On souhaite affecter l'adresse de de X dans une variable de type pointeur. Soient les instructions suivantes :

```
1)  int *PX ;
2)   PX = &X ;
```

Dans l'instruction 1, vous constatez, lors de la déclaration de la variable PX, on l'a fait précéder par * (étoile ou Astérix). Cette notation, c'est-à-dire, faire précéder une variable par * permet de déclarer une variable qui va contenir l'adresse d'une autre variable. Ce genre de variable s'appelle POINTEUR.

Dans l'instruction 2 PX = &X ; cette instruction permet d'affecter l'adresse de X dans la variable PX. On dit que PX pointe sur X.

Voici un exemple schématisant un pointeur PX qui pointe sur une variable X.

On accède à la donnée pointée par un pointeur (valeur de x dans l'exemple précédent) par une étoile.

Pour accéder à la valeur d'une variable pointée par un pointeur, on utilise l'instruction suivante :

```
printf("%d", *PX);
```

Dans cette instruction, nous appelons la fonction printf qui va afficher le contenu de X et non le conte de PX. Vous constatez que le pointeur PX est précédé par une étoile, ce qui signifie la valeur de la variable pointée par PX qui est X.

Lors de la déclaration d'un pointeur, il faut faire précéder la variable pointeur par une étoile.

Lors de l'accès à la valeur de la variable pointée par un pointeur, vous faites précéder le pointeur par une étoile.

Soit l'exemple suivant :

```
1) int X, Y, Z
2) int *PX
3) PX= &X
4) X = 20
5) Y=*PX
6) PX=&Z
```

Dans l'instruction1, Nous déclarons trois variables de type entiers X, Y, Z.

Dans l'instruction 2, nous déclarons une variable pointeur PX

Dans l'instruction 3, nous affectons l'adresse de X à PX, c'est-à-dire que le pointeur PX pointe sur X.

Dans l'instruction 4, nous affectons la valeur 20 à la variable X.

Dans l'instruction5, nous affectons la valeur de la variable pointée par PX à Y. La variable pointée est X et son contenu est 20 qui sera affectée à la variable Y.

Dans l'instruction 6, nous affectons l'adresse de Z au pointeur PX. C'est-à-dire que le pointeur pointe maintenant sur Z et non sur X.

Soit le tableau suivant, récapitulant l'exécution des 6 instructions ci-dessus :

Instruction 1 : int X, Y, Z	X	Y	Z	
Instruction 2 : int *PX	X	Y	Z	PX
Instruction 3 : PX= &X				&X
Instruction 4 : X = 20	20			
Instruction 5 : Y=*PX	20	20		
Instruction 6 : PX=&Z	20	20		&Z

Ne pas confondre l'usage de l'étoile lors de la déclaration d'une variable de type pointeur avec l'usage de l'étoile qui permet d'accéder à l'objet pointé par le pointeur.

Int *p ; // ici nous déclarons un pointeur de type entier.

Int x ;

p=&x ;

*p = 20 ; // ici nous affectons 20 à x et non à p. *p désigne la valeur de la variable pointée par p qui est x.

En résumé, lorsque p pointe sur x, la valeur de p est l'adresse de x, toute modification de *p modifie x et toute modification de x modifie *p. La raison est que *p et x sont sur le même emplacement mémoire dans la mémoire RAM.

Opérations sur les pointeurs

1) int x,y;
2) int *p;
3) p= &x;
4) *p=125;
5) y=*p+10;
6) y=++(*p);
7) y=(*p)++;

L'instruction 1 : int x,y

Nous déclarons deux variables entières x et y.

L'instruction 2 : int *p

Nous déclarons un pointeur de type entier : p.

L'instruction 3 : p= &x

Nous affectons l'adresse de x au pointeur p. Le pointeur p pointe maintenant sur x.

L'instruction 4 :*p=125

*p désigne la variable x, c'est-à-dire nous affectons à x la valeur 125.

L'instruction 5 :y=*p+10

Nous affectons à y la somme de *p qui est la valeur de x, donc 125 à 10. La variable y a la valeur 135.

L'instruction 6 : y=++(*p)

La valeur de *p c'est-à-dire la valeur de x sera incrémentée de 1. Donc la valeur de x sera 126 puis cette valeur qui 126 sera affectée à y. Ainsi x a la valeur 126 et y a la valeur 126. Vous pouvez écrire y = ++ *p c'est-à-dire sans utiliser les parenthèses. Mais je vous conseilles des les utiliser.

L'instruction 7 : y=(*p)++

La variable y reçoit la valeur de *p c'est-à-dire la valeur de x donc y aura la valeur 126. Puis *p sera incrémentée de 1. Donc la variable x aura la valeur 127. Ici il faut mettre les parenthèses pour enlever toute ambiguïté au compilateur.

Soit le tableau suivant récapitulant les 7 instructions :

L'instruction 1 : int x,y	x	y	
L'instruction 2 : int *p	x	y	p
L'instruction 3 : p= &x			&x
L'instruction 4 :*p=125	125		&x
L'instruction 5 :y=*p+10	125	135	&x
L'instruction 6 : y=++(*p)	126	126	&x
L'instruction 7 : y=(*p)++	127	126	

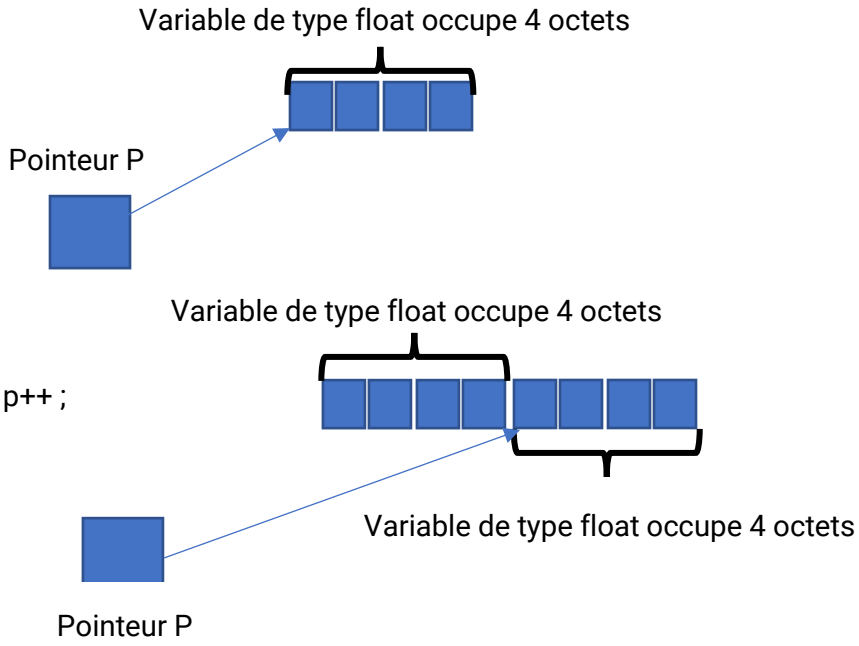
Voici un programme C pour tester ces instructions :

```
#include <stdio.h>


int main()
{
	int x,y;
	int *p;
	p= &x;
	*p=125;
	printf("x = %d\n",*p);
	y=*p+10;
	printf("y = %d\n",y);
	y=++(*p);
	printf("x = %d\n",*p);
	printf("y = %d\n",y);
	y=*p++;
	printf("x = %d\n",*p);
	printf("y = %d\n",y);
	return 0;
}
```

Quelle différence entre (*p)++ et p++ ?

- (*p)++ désigne que le contenu de la variable pointée par sera incrémentée de 1.
- P++ désigne que l'adresse contenue dans le pointeur va changer et va recevoir l'adresse actuelle plus 4 octets suivants dans le cas d'un float par exemple.



Au début, p pointe sur le premier octet d'une variable de type float. C'est-à-dire qu'il contient l'adresse du premier octet de la variable de type float.

L'instruction p++ ; permet l'incrémentation de l'adresse contenue dans le pointeur p. c'est-à-dire que le pointeur va contenir l'adresse de l'octet 5. Le pointeur pointe sur l'octet 5 c'est-à-dire le début d'une autre variable.