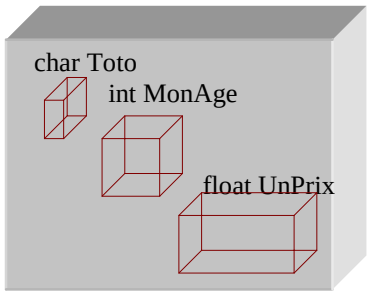
Qu'est-ce qu'un pointeur ?

Imaginons un très simple programme qui commence par la déclaration de trois variables, cette déclaration ayant pour but de réserver l'espace nécessaire au storage des données. Chaque variable obtient un espace mémoire localisé là où il y a de la place, un peu comme les usagers d'un autobus prennent les sièges restant. C'est premier arrivé premier servi.

Exécution du programme

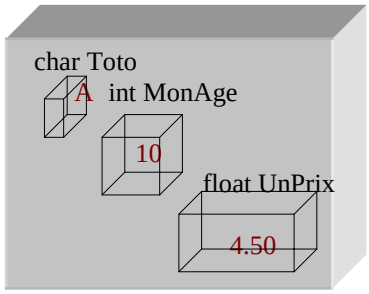


Code du programme

```
char Toto;
int MonAge;
float UnPrix;
```

Comme à l'habitude, nous effectuerons l'initialisation de nos variables en leur assignant une valeur. Pour cela, nous utilisons l'opérateur = qui transforme son opérande de gauche avec le résultat reçu du coté droit.

Exécution du programme



Code du programme

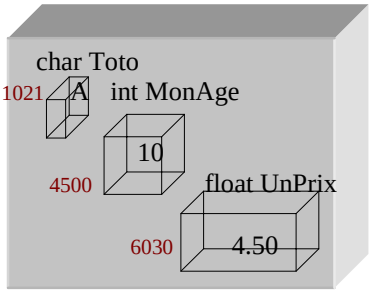
```
char Toto;
int MonAge;
float UnPrix;

Toto = 'A';
MonAge = 10;
UnPrix = 4.50;
```

Actuellement dans ce code, la seule façon de modifier ces variables est d'utiliser explicitement leur nom. Imaginez si tous les programmes étaient implémentés de cette façon, il serait difficile, voire impossible, de traiter des masses de données, de les classer, d'effectuer des recherches, etc. Heureusement, il existe un opérateur pour nous donner l'adresse en mémoire d'une variable, un peu comme on obtiendrait le numéro de siège d'un passager de l'autobus.

L'opérateur de déréférence & (ou l'opérateur d'adresse)

Imaginons que les adresses de nos variables sont respectivement 1021, 4500 et 6030, l'opérateur & appliqué à ces variables retournera ces adresses. À présent, nous avons besoin de mémoriser, storer ces adresses pour pouvoir les utiliser plus tard.

Exécution du programme



Code

```
&Toto donne 1021
&MonAge donne 4500
&UnPrix donne 6030
```

Le type pointeur

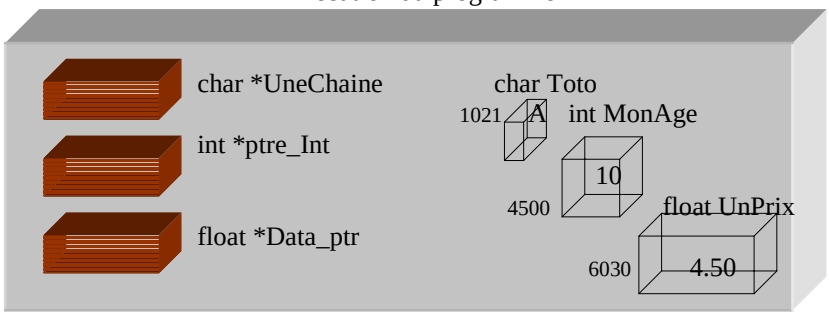
Le type pointeur indique qu'une variable va contenir l'adresse d'une autre variable. Un peu comme dans une chasse au trésor, le programme devra regarder à l'intérieur de la variable pointeur l'adresse de la prochaine variable. On dit que le pointeur pointe vers la variable.

Il est même possible d'avoir un pointeur qui contient l'adresse d'un autre pointeur, qui lui contient l'adresse d'une variable normale. Le premier est donc un double pointeur et pour retrouver les données, il faut encore utiliser le principe de la chasse au trésor.

La syntaxe pour déclarer un pointeur est de nommer le type de la variable puis une * puis le nom qu'on lui donne. Par exemple :

```
char *UneChaine;
int *ptr_Int;
float *Data_ptr;
```

Exécution du programme



Code du programme

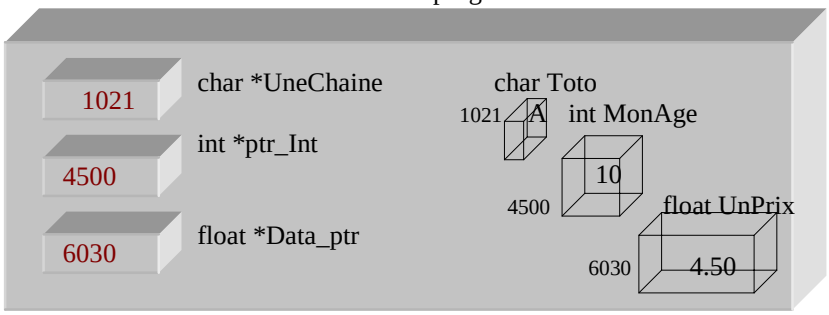
```
char Toto = 'A';
int MonAge = 10;
float UnPrix = 4.50;
char *UneChaine;
int *ptr_Int;
float *Data_ptr;
```

Un type pointeur va contenir l'adresse d'une variable, on peut donc dire que l'adresse d'une variable est de type pointeur. Déjà, dès l'application de l'opérateur de déréférence &, le nombre obtenu est de type pointeur à la variable. Par exemple, supposons que MaLettre est de type char, si j'écris &MaLettre j'obtient quelque chose de type char *. Ce quelque chose est bien sur un nombre représentant l'adresse en mémoire de la variable MaLettre.

Il suffit donc d'assigner au pointeur l'adresse obtenue avec l'opérateur & comme ceci :

```
char *unPointeur;
unPointeur = &MaLettre;
```

Exécution du programme



Code du programme

```
char Toto = 'A';
int MonAge = 10;
float UnPrix = 4.50;
char *UneChaine;
int *ptr_Int;
float *Data_ptr;
UneChaine = &Toto;
ptr_Int = &MonAge;
Data_ptr = &UnPrix;
```

Maintenant, il ne reste plus qu'à apprendre à accéder à la valeur d'une variable régulière à partir d'un pointeur qui contient son adresse, sans nommer cette variable.

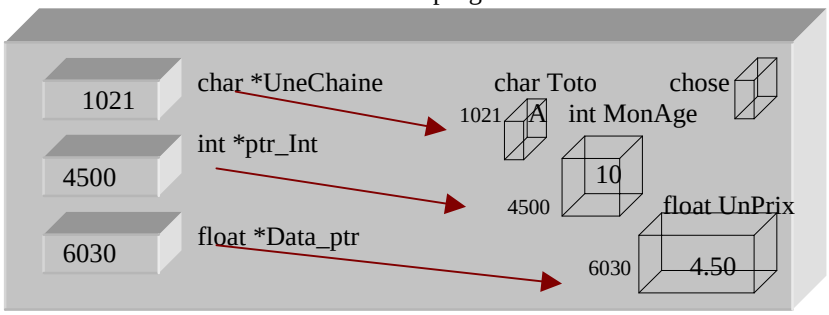
L'opérateur de référence *

Lorsqu'appliqué à la gauche d'une variable de type pointeur, l'opérateur * lit l'adresse storé par le pointeur, va à la location mémoire correspondante et accède le contenu de cette location en mémoire, l'opération retourne ou modifie le contenu de la variable pointée. Par exemple, *UnPointeur accède le contenu de MaLettre.

Bien sur, si le référencement a lieu à la droite d'un opérateur =, la valeur de la variable est donnée (retournée) et si le référencement a lieu à la gauche d'un =, le contenu de la variable pointée est modifié.

qqchose = *UneChaine; ici qqchose devient 'A'
*UneChaine = 'B'; ici Toto devient B

Exécution du programme

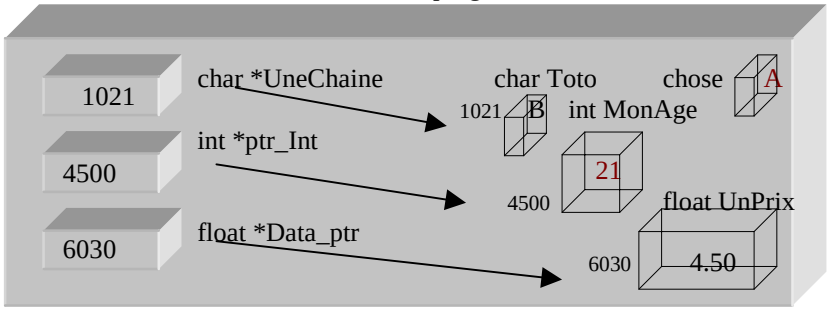


Code du programme

```
char chose, Toto = 'A';
int MonAge = 10;
float UnPrix = 4.50;
char *UneChaine;
int *ptr_Int;
float *Data_ptr;
UneChaine = &Toto;
ptr_Int = &MonAge;
Data_ptr = &UnPrix;
```

*UneChaine accède Toto
*ptr_Int accède MonAge
*Data_ptr accède UnPrix

Exécution du programme



Code du programme

```
char Toto = 'A';
int MonAge = 10;
float UnPrix = 4.50;
char *UneChaine;
int *ptr_Int;
float *Data_ptr;
UneChaine = &Toto;
ptr_Int = &MonAge;
Data_ptr = &UnPrix;

chose = *Toto;
*Toto = 'B';
scanf("%d", ptr_Int );
*ptr_Int = 21;
```

*UneChaine accède Toto
*ptr_Int accède MonAge
*Data_ptr accède UnPrix

chose devient 'A'
Toto devient 'B'
On lit MonAge en utilisant directement le pointeur, sans &, car il est déjà l'adresse de la variable
MonAge devient 20

N.B.: Ne pas confondre l'opérateur de référencement * avec l'étoile utilisée lors de la déclaration du type pointeur. Par exemple, il peut arriver que la déclaration et l'initialisation soient en une seule instruction.

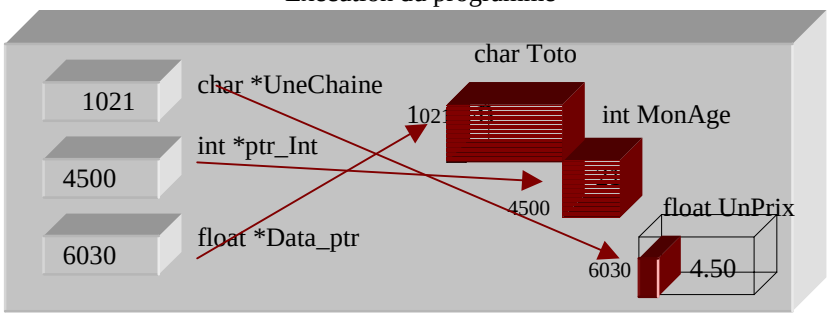
Si un pointeur de type (char *) est référencé avec l'opérateur *, le type devient char. Par exemple *ptr_char a un type global de char.

À l'opposé, même s'il y a une étoile devant le nom de la variable au moment de la déclaration, celle-ci fait partie de l'énoncé du type et n'effectue pas l'opération de référencement.
int *ptr = &variable; peut être décomposé en int *ptr; ptr = &variable;
int *ptr2 =ptr; peut être décomposé en int *ptr2; ptr2 = ptr1;
Il faut lire ces déclarations ainsi: (int *)ptr = &variable; (int *)ptr2 =ptr;
Ici ptr et ptr2 ont un type (int *) et doivent donc recevoir une valeur de type (int *) telle que l'adresse d'un int avec l'opération &de_la_variable_int ou encore recevoir directement un autre pointeur de type (int *) également.

Pourquoi un pointeur char * vers une variable de type char et un int * vers un int?

Parce que l'opération de référencement doit bien se passer, récupérer tout l'espace mémoire de la variable mais pas plus sinon il y a danger de segfault (accès à une zone de mémoire non autorisée causant la terminaison immédiate du programme). La façon de dire au compilateur combien de mémoire accéder est de donner au pointeur le même type que la variable pointée.

Exécution du programme



*UneChaine accède UnPrix
*ptr_Int accède MonAge
*Data_ptr accède Toto

Code du programme

```
char Toto = 'A';
int MonAge = 10;
float UnPrix = 4.50;
char *UneChaine;
int *ptr_Int;
float *Data_ptr;

UneChaine = &UnPrix;
ptr_Int = &MonAge;
Data_ptr = &Toto;

*UneChaine=      ;
*ptr_Int=        ;
*Data_ptr=       ;
```

*UneChaine est réglé pour accéder à l'espace d'un char mais accède à un énorme float : UnPrix, le résultat correspond au premier octet de la variable UnPrix, ce n'est probablement pas le résultat escompté ! Cela ne cause pas de débordement de mémoire, au contraire, on n'en prend pas assez. *ptr_Int accède MonAge complètement et sans déborder. *Data_ptr qui est réglé pour accéder à l'espace d'un float (car Data_ptr est de type float *) mais qui accède un tout petit char écrira dépassé l'espace de la variable char, sur de la mémoire censée servir à autre chose… c'est une grave erreur.

Un autre usage du type des pointeurs sera étudié ultérieurement, dans la prochaine section : Tableaux de variables et arithmétique des pointeurs.