Sommaire

Prérequis

1. Comprendre ce qu'est une variable, et savoir les manipuler.
2. Savoir déclarer un tableau et l'utiliser.
3. Savoir manipuler différentes fonctions usuelles.
4. Savoir concevoir et manipuler des fonctions.
5. Être à l'aise avec la portée des variables.

Enseignement des pointeurs

Un outil difficile à maitriser

La notion de pointeur :

1. Est connue pour poser des difficultés aux étudiants.
2. Augmente fortement le risque d'erreurs de programmation (et donc de plantages) si on manque de rigueur.
   1. Mars, le rover Curiosity planté.
   2. Le premier vol d'Ariane 5 est un échec. La fusée explose. Une erreur à 160 000 000 d'euros.

Enseignement des pointeurs

Pourquoi étudier les pointeurs ?

De nombreux langages semblent s'abstraire des pointeurs en ne proposant pas leur manipulation. Mais :

1. Reflètent la manière dont est conçue et gérée la mémoire.
2. Permettent de manipuler de manière précise la mémoire :
   1. Programmation système.
   2. Programmes courts, efficaces et rapides.
   3. Gestion très fine de la mémoire occupée.
   4. Très formateur, algorithmiquement parlant.

Enseignement des pointeurs

Une notion servant d'indicateur

Une étude montre que :

- Les enseignants passent beaucoup de temps à enseigner les pointeurs.
- 70% des développeurs déclarent avoir compris cette notion en travaillant seuls, chez eux.

Sommaire

La mémoire, une suite d'octets

La mémoire d'un ordinateur peut-être vue comme une succession d'octets :

1. Un octet contient 8 bits. Chaque bit peut valoir 0 ou 1.
2. Chaque octet possède une adresse.
3. Les adresses commencent à 1.
4. Les adresses croissent de 1 en 1.
5. L'usage veut que ces adresses soient données en hexadécimal (base 16: 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F).

Déclaration de variable

Lors de la déclaration d'une variable :

1. Un espace mémoire est réservé (alloué) automatiquement.
2. La taille de cet espace dépends de son type :
3. Le nom de la variable est lié à cet emplacement mémoire.

Nombre d'octets d'une variable

L'opérateur sizeof

Le nombre d'octets utilisés pour stocker une variable est donné par l'opérateur sizeof :

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    printf("sizeof(void)          = %d\n", sizeof(void));
    printf("sizeof(char)          = %d\n", sizeof(char));
    printf("sizeof(short)         = %d\n", sizeof(short));
    printf("sizeof(int)           = %d\n", sizeof(int));
    printf("sizeof(long)          = %d\n", sizeof(long));
    printf("sizeof(long long)     = %d\n", sizeof(long long));
    printf("sizeof(float)         = %d\n", sizeof(float));
    printf("sizeof(double)        = %d\n", sizeof(double));
    return EXIT_SUCCESS;
}
```

```
sizeof(void)          = 1
sizeof(char)          = 1
sizeof(short)         = 2
sizeof(int)           = 4
sizeof(long)          = 4
sizeof(long long)     = 8
sizeof(float)         = 4
sizeof(double)        = 8
```

Attention ! Ces tailles sont données à titre d'exemple et ne constituent pas une vérité absolue. Elles dépendent de plusieurs facteurs (processeur, système d'exploitation, version du compilateur).

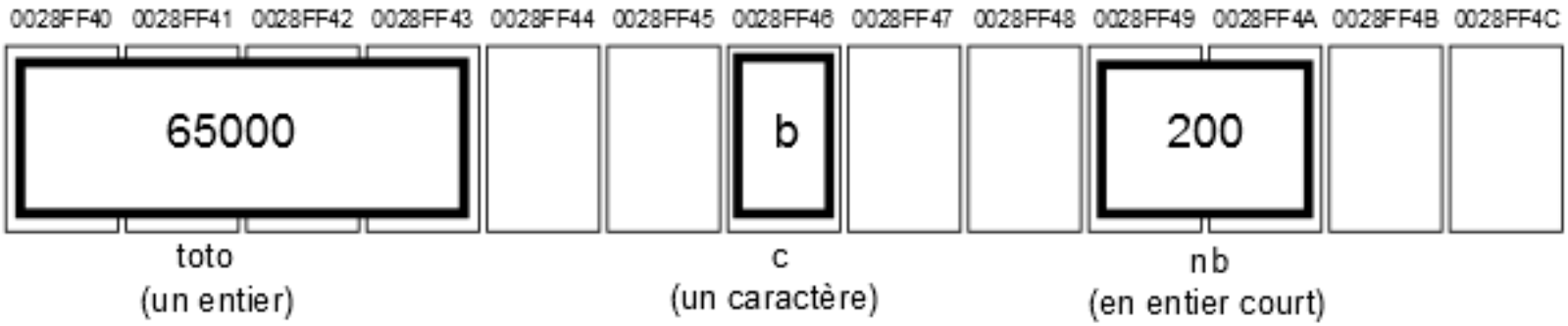
Déclaration de variables

Exemple

Le code suivant va déclarer 3 variables, et les initialiser :

```
1  int toto;
2  short nb;
3  char c;
4
5  toto=65000;
6  nb=200;
7  c='b';
```

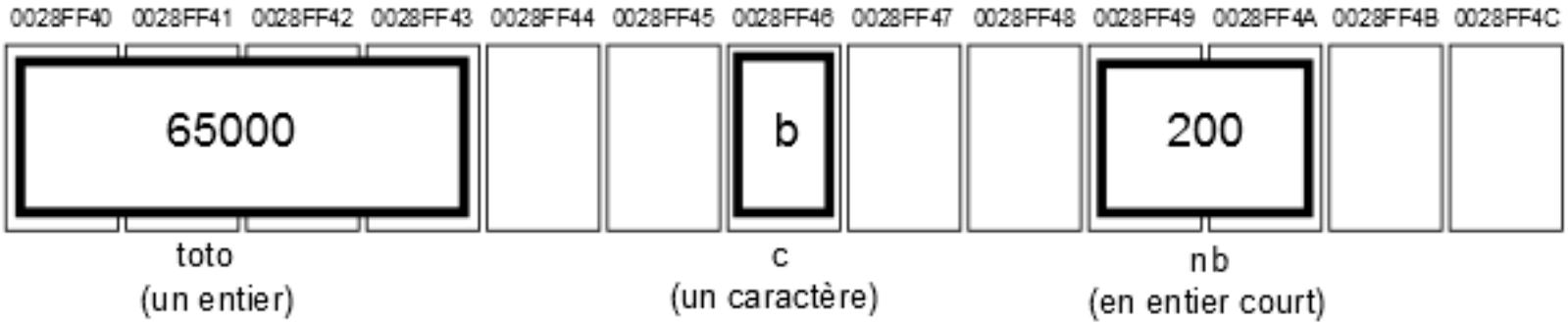
Voici l'effet produit en mémoire :



Déclaration de variables

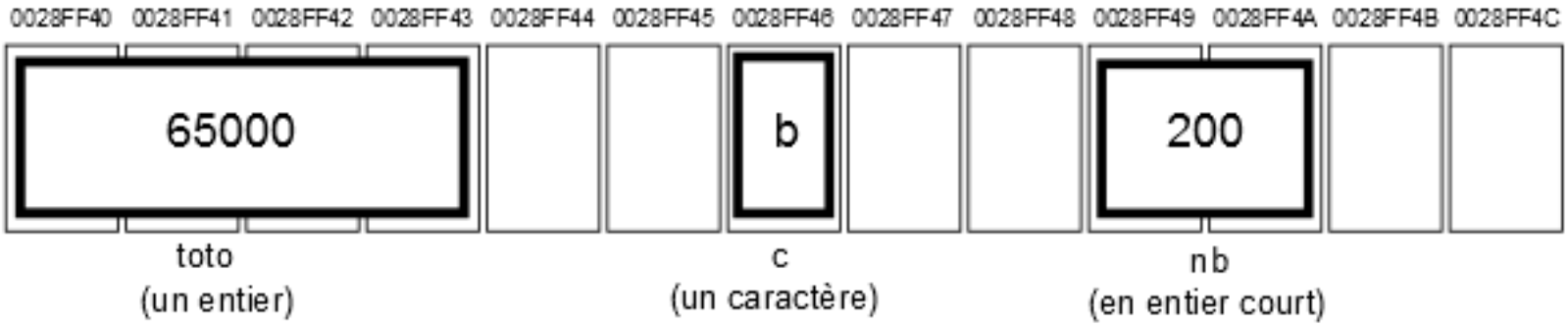
Adresse mémoire d'une variable

D'après vous ? Quelle sont les adresses des variables toto, c et nb ?



Déclaration de variables

Adresse mémoire d'une variable



1. L'adresse d'une variable correspond à l'adresse du premier de ses octets :
   - toto : 0028FF40
   - c : 0028FF46
   - nb : 0028FF49
2. On connait le type de la variable, donc sa taille.

Déclaration de variables

Adresse mémoire d'une variable

Comment obtenir l'adresse d'une variable directement depuis un programme ?

L'opérateur unaire &

L'opérateur unaire & placé devant le nom d'une variable retourne l'adresse de cette variable.

Déclaration de variables

Adresse mémoire d'une variable

Pour afficher l'adresse retournée en hexadecimal, on peut utiliser le format %p.

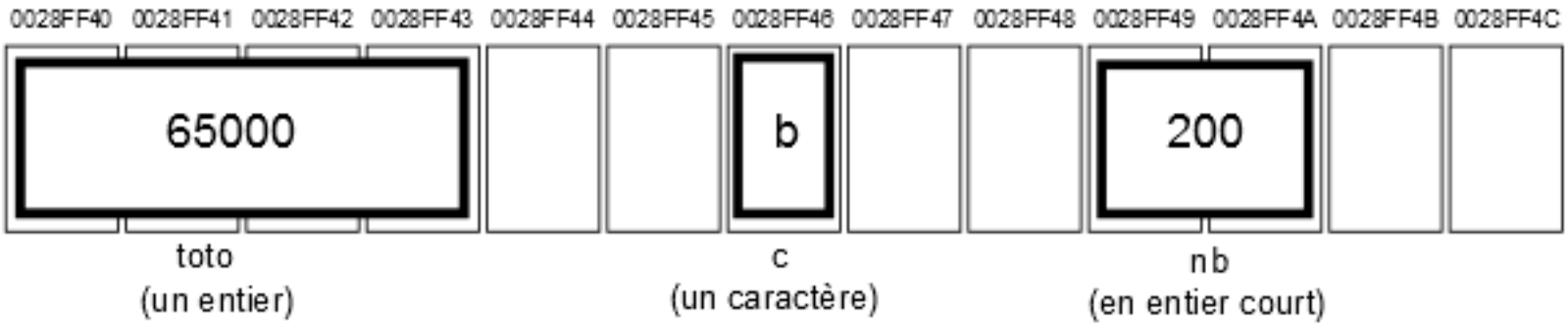
```c
#include<stdio.h>
#include<stdlib.h>

int main()
{
    int x=1;
    printf("La variable x a pour valeur : %d\n",x);
    printf("La variable x a pour adresse : %p\n",&x);
    return EXIT_SUCCESS;
}
```

```
La variable x a pour valeur : 1
La variable x a pour adresse : 0028FF44
```

Déclaration de variables

Valeur ≠ adresse

Il faut bien distinguer la valeur d'une variable, c'est à dire ce que l'on enregistre dans la variable, de l'adresse de la variable, c'est à dire de l'endroit où cette valeur est enregistrée.



Retour sur la fonction scanf

Vous comprenez mieux ce qui se passe ?

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int x=1;
    printf("Entrez une valeur : ");
    scanf("%d",&x);
    return 0;
}
```

Sommaire

Les pointeurs

Si vous avez compris :

1. Comment déclarer une variable.
2. Comment sont enregistrées les variables en mémoire.
3. Manipuler le contenu d'une variable grâce à l'opérateur d'affectation '='.
4. Comment récupérer l'adresse d'une variable.

Alors nous pouvons entrer dans le vif du sujet.

Les pointeurs

Pointeur

Un pointeur est une variable dont la valeur est égale à une adresse mémoire. En particulier, cette adresse peut correspondre à l'adresse d'une autre variable. On dit alors que le pointeur « pointe » vers cette variable.

Exemple de déclaration d'un pointeur :

```
1   type *nom_du_pointeur;
```

avec type le type de la variable vers laquelle on pointe.

Les pointeurs

Exemple d'utilisation

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int *p;
    int x=1;
    printf("La variable x a pour valeur : %d\n",x);
    printf("La variable x a pour adresse : %p\n",&x);
    p=&x;
    printf("Le pointeur p a pour adresse : %p\n",&p);
    printf("Le pointeur p a pour valeur : %p\n",p);
    return 0;
}
```

```
La variable x a pour valeur : 1
La variable x a pour adresse : 0028FF40
Le pointeur p a pour adresse : 0028FF44
Le pointeur p a pour valeur : 0028FF40
```

Les pointeurs

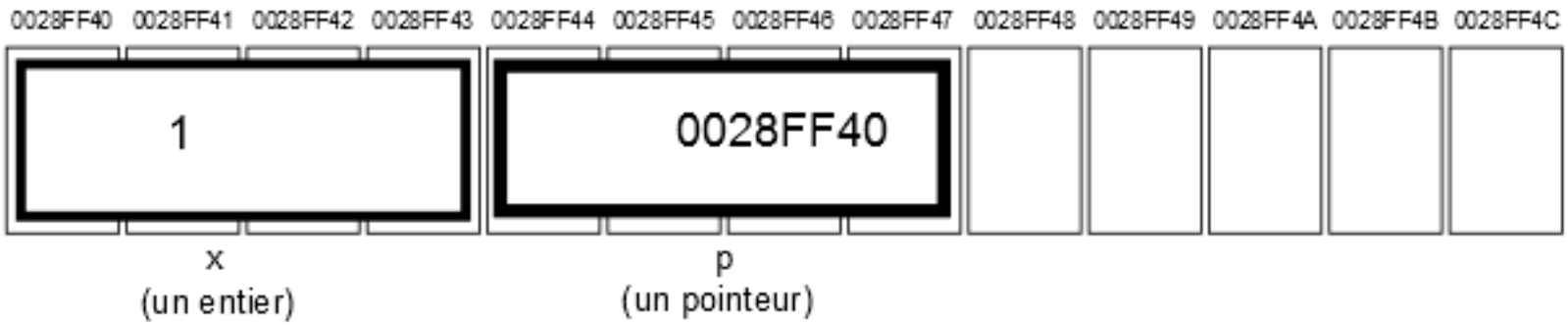
Exemple d'utilisation. Explications.

1. **ligne 6** : on déclare p, un pointeur d'entier.
2. **ligne 7** : on déclare une variable entière x, que l'on initialise à 1.
3. **ligne 8** : on affiche la valeur contenue dans la variable x.
4. **ligne 9** : on affiche l'adresse de la variable x.
5. **ligne 10** : on fait pointer p vers x, en affectant l'adresse de x à p. Concrètement, la valeur contenue dans la variable p sera l'adresse de la variable x.
6. **ligne 11** : on affiche l'adresse de la variable p. En effet, p est une variable (un pointeur certes, mais une variable tout de même). Comme toute variable, elle est stockée en mémoire à une certaine adresse, celle que l'on affiche donc.
7. **ligne 12** : on affiche le contenu de la variable p, c'est à dire l'adresse de la variable x.

Les pointeurs

Exemple d'utilisation. Représentation en mémoire.

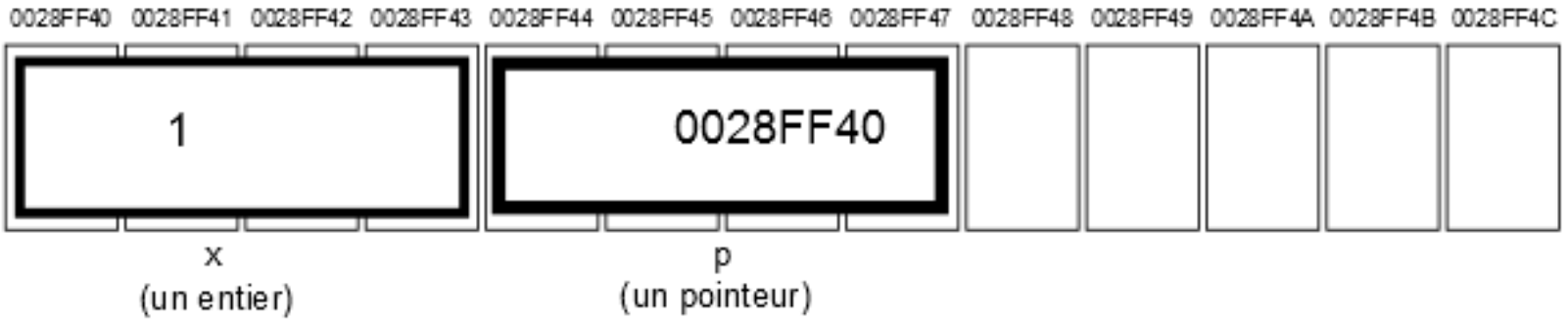
```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int *p;
    int x=1;
    printf("La variable x a pour valeur : %d\n",x);
    printf("La variable x a pour adresse : %p\n",&x);
    p=&x;
    printf("Le pointeur p a pour adresse : %p\n",&p);
    printf("Le pointeur p a pour valeur : %p\n",p);
    return 0;
}
```



Les pointeurs

L'opérateur d'indirection

Étant donné l'adresse d'une variable, comment accéder à la valeur située à cette adresse ?



Autrement dit, comment lire la valeur 1 à partir de p ?

Les pointeurs
L'opérateur d'indirection

Pour modifier un contenu situé à une adresse donnée, il faut utiliser l'opérateur d'indirection :

L'opérateur d'indirection *

L'opérateur unaire *, aussi appelé opérateur d'indirection, permet d'accéder à la valeur stockée à l'adresse qui est contenue dans un pointeur. Tout comme l'opérateur &, l'opérateur * est préfixé.

Les pointeurs

L'opérateur d'indirection

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int *p;
    int x=1;

    printf("La variable x a pour valeur : %d\n",x);
    printf("La variable x a pour adresse : %p\n",&x);

    p=&x;
    *p=5;

    printf("La variable x a pour valeur : %d\n",x);
    printf("La variable x a pour adresse : %p\n",&x);

    return 0;
}
```

```
La variable x a pour valeur : 1
La variable x a pour adresse : 0028FF40
La variable x a pour valeur : 5
La variable x a pour adresse : 0028FF40
```

Sommaire

Initialisation des pointeurs
La constante NULL

Il est fortement recommandé d'initialiser les variables lors de leur déclaration. Pour un entier, il suffit de lui donner une valeur comme 0, 1 ou toute autre valeur ayant une signification à vos yeux.

La constante NULL

La constante NULL est définie comme étant une adresse mémoire qui n'existe pas. L'adressage de la mémoire commençant à 1, il est d'usage de définir cette constante à 0:

```
1  #define NULL ((void *)0)
```

1. Un pointeur initialisé à NULL signifie qu'il ne pointe vers rien.
2. Cette convention permet de déterminer si un pointeur à déjà été manipulé depuis sa déclaration, de détecter des erreurs.
3. Pensez à bien initialiser vos pointeurs, c'est important.

Sommaire

Déclaration d'un pointeur

Pourquoi typer un pointeur ?

Si vous avez tout compris jusqu'ici, vous devriez vous poser une question fondamentale. Reprenons la déclaration d'un pointeur :

Déclaration d'un pointeur

La déclaration d'un pointeur se fait de la manière suivante :

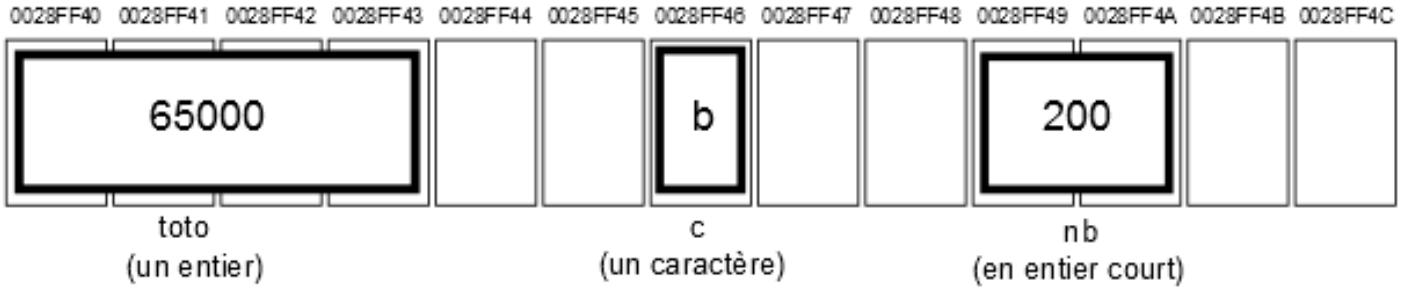
```
1 type * nom_du_pointeur;
```

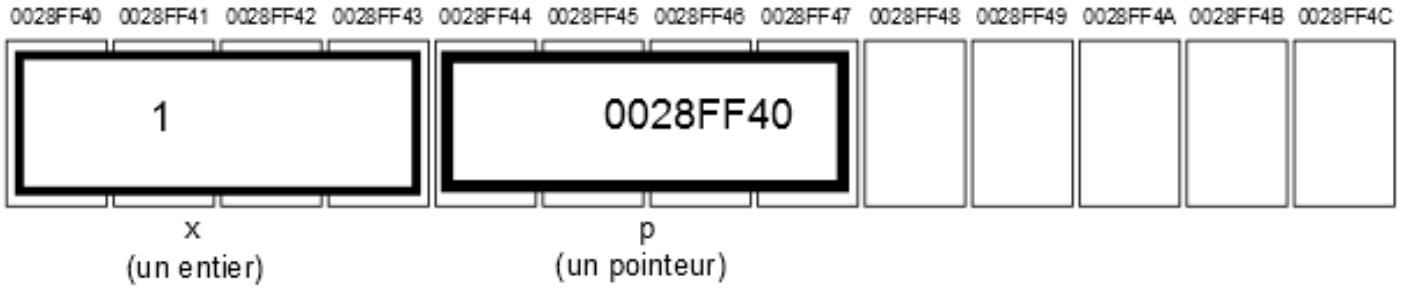
avec type le type de la variable vers laquelle on pointe.

Déclaration d'un pointeur

Pourquoi typer un pointeur ?

Or, tous les octets de la mémoire possèdent une adresse. Cette adresse est toujours de la même forme :





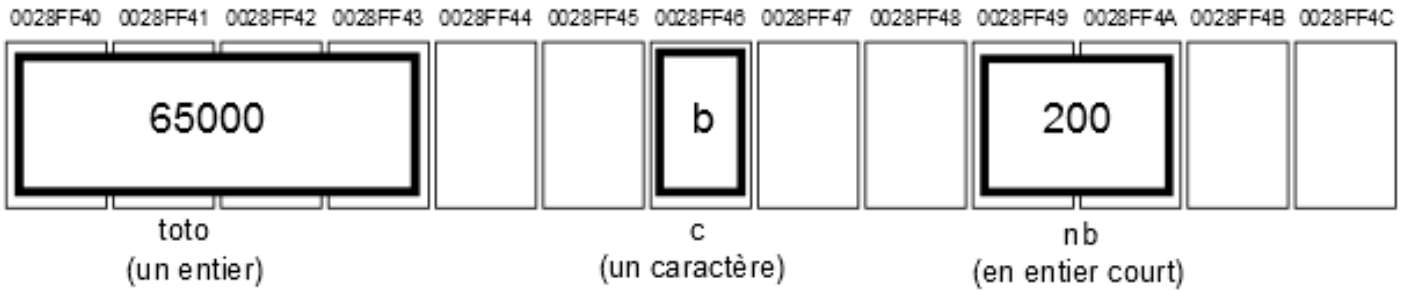
Donc, le type d'un pointeur devrait être "adresse" ou quelque chose de similaire...

Déclaration d'un pointeur

Pourquoi typer un pointeur ?

```
1  int toto = 65000;
2  char c = 'b';
3  short nb = 200;
4
5  char *p;
6  p = &toto;
```

Nous avons donc en mémoire les trois variables toto,c et nb, ainsi qu'un pointeur p de type char (non présent sur l'image) :



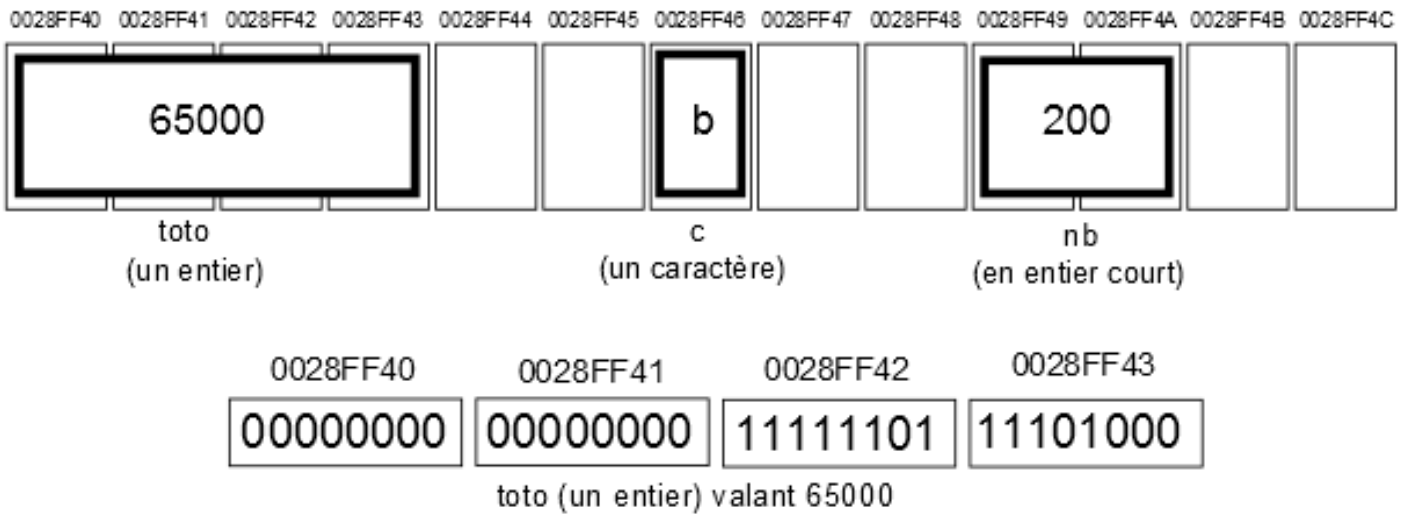
Que se passe-t-il si nous executons ensuite l'instruction suivante :

```
1  *p='A';
```

Déclaration d'un pointeur

Pourquoi typer un pointeur ?

Voici l'état de la mémoire AVANT l'exécution de l'instruction :

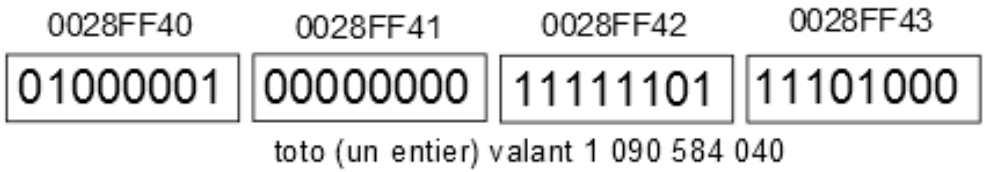


Déclaration d'un pointeur

Pourquoi typer un pointeur ?

Voici l'état de la mémoire AVANT l'exécution de l'instruction :

Voici l'état de la mémoire APRES l'exécution de l'instruction :



Maintenant, lorsque l'on va lire la variable toto, elle aura pour valeur 1 090 584 040. Plus aucun rapport avec la valeur de 65000, ni la lettre 'A'.

Déclaration d'un pointeur

Pourquoi typer un pointeur ?

Lorsqu'on affecte une valeur, combien d'octets doit-on écrire ?

1. Les pointeurs doivent avoir un type en lien avec le type de données vers laquelle ils pointent.
2. Le type d'un pointeur permet de savoir comment écrire une donnée à l'adresse pointée (nombre d'octets, ...)

Quelles sont les adresses des cases d'un tableau ?

Quel sera l'affichage de ce programme ?

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int i;
    int mon_tableau[10];
    for(i=0;i<10;i++)
    {
        printf("L'adresse de la case %d est %p\n",i,&(mon_tableau[i]));
    }
    return 0;
}
```

Quelles sont les adresses des cases d'un tableau ?

Quel sera l'affichage de ce programme ?

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int i;
    int mon_tableau[10];
    for(i=0;i<10;i++)
    {
        printf("L'adresse de la case %d est %p\n",i,&(mon_tableau[i]));
    }
    return 0;
}
```

```
L'adresse de la case 0 est 0060FEE4
L'adresse de la case 1 est 0060FEE8
L'adresse de la case 2 est 0060FEEC
L'adresse de la case 3 est 0060FEF0
L'adresse de la case 4 est 0060FEF4
L'adresse de la case 5 est 0060FEF8
L'adresse de la case 6 est 0060FEFC
L'adresse de la case 7 est 0060FF00
L'adresse de la case 8 est 0060FF04
L'adresse de la case 9 est 0060FF08
```

Quelles sont les adresses des cases d'un tableau ?

Petit rappel

1. Un tableau est un ensemble de variables de même type, stockées en mémoire de manière contiguës.
2. Chaque variable d'un même type occupe le même nombre d'octets en mémoire
3. La mémoire de l'ordinateur est adressée octet par octet

Quelles sont les adresses des cases d'un tableau ?

Explication sur l'affichage du programme

1. Affichage de l'adresse du premier élément du tableau, donc de la première case.
2. Si le type des éléments du tableau occupe sizeof(type) octets, chaque case du tableau va occuper sizeof(type) octets.
3. Comme les cases sont placées les unes à la suite des autres, chaque case se trouve donc sizeof(type) octets plus loin que la précédente.

Quelles sont les adresses des cases d'un tableau ?

De quoi avons nous besoin pour accéder à la valeur d'une case d'un tableau ?

Jusqu'à présent :

```
1  int tableau[10];
2  tableau[4]=8;
```

1. Le nom du tableau.
2. Le type des données stockées dans le tableau.
3. Le numéro de la case.

Mais en fait, nous avons besoin de :

1. L'adresse de la première case.
2. Le type des données stockées dans le tableau.
3. Le numéro de la case.

Quelles sont les adresses des cases d'un tableau ?

Révélation

Phrase magique

Le nom d'un tableau est un pointeur sur son premier élément.

C'est à dire par exemple que l'expression:

```
1  &(mon_tableau[0])
```

vaut la même chose que l'expression

```
1  mon_tableau
```

Quelles sont les adresses des cases d'un tableau ?

Exemple d'affectation

Si on souhaite affecter une valeur dans la première case d'un tableau :

```
1  mon_tabeau[0]=5;
```

Ou bien, en utilisant le fait que le nom d'un tableau est un pointeur sur son premier élément :

```
1  *mon_tabeau=5;
```

Arithmétique des pointeurs et sucre syntaxique

Comment réaliser une affectation dans une autre case du tableau ?

Si on souhaite affecter une valeur dans la deuxième case d'un tableau :

```
1   mon_tabeau[1]=5;
```

Mais comment faire en utilisant les pointeurs ?

Arithmétique des pointeurs et sucre syntaxique

Comment réaliser une affectation dans une autre case du tableau ?

Si on souhaite affecter une valeur dans la deuxième case d'un tableau :

```
1  mon_tabeau[1]=5;
```

Mais comment faire en utilisant les pointeurs ? La logique voudrait que l'on écrive :

```
1  *(mon_tabeau+sizeof(int))=5;
```

Arithmétique des pointeurs et sucre syntaxique

Comment réaliser une affectation dans une autre case du tableau ?

Malheureusement (ou heureusement ?), les choses ne fonctionnent pas de cette manière. Le typage des pointeurs permet au compilateur de connaitre la taille des données et le langage C va utiliser cette propriété pour nous simplifier la vie. Voici la bonne instruction :

```
1   *(mon_tabeau+1)=5;
```

Quand on souhaite accéder à l'adresse mon_tableau+ 1, le compilateur comprend que l'on souhaite accéder à l'élément situé 1 case après l'adresse du premier élément mon_tableau soit sizeof(int) octets plus loin. Pratique non ?

Arithmétique des pointeurs et sucre syntaxique

Comment réaliser une affectation dans une autre case du tableau ?

```c
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int i;
    int mon_tableau[10];
    for(i=0;i<10;i++)
    {
        printf("L'adresse de la case %d est %p qui vaut bien %p\n",i,&(
            mon_tableau[i]),mon_tableau+i);
    }
    return 0;
}
```

```
L'adresse de la case 0 est 0060FEE4 qui vaut bien 0060FEE4
L'adresse de la case 1 est 0060FEE8 qui vaut bien 0060FEE8
L'adresse de la case 2 est 0060FEEC qui vaut bien 0060FEEC
L'adresse de la case 3 est 0060FEF0 qui vaut bien 0060FEF0
L'adresse de la case 4 est 0060FEF4 qui vaut bien 0060FEF4
L'adresse de la case 5 est 0060FEF8 qui vaut bien 0060FEF8
L'adresse de la case 6 est 0060FEFC qui vaut bien 0060FEFC
L'adresse de la case 7 est 0060FF00 qui vaut bien 0060FF00
L'adresse de la case 8 est 0060FF04 qui vaut bien 0060FF04
L'adresse de la case 9 est 0060FF08 qui vaut bien 0060FF08
```

Arithmétique des pointeurs et sucre syntaxique

Comment réaliser une affectation dans une autre case du tableau ?

Il y a équivalence entre les deux notations suivantes :

```
1  mon_tableau[i]
```

et

```
1  *(mon_tableau+i)
```

C'est ce qu'on appelle le sucre syntaxique. Le sucre, c'est bon. Donc l'une de ces deux notations est plus agréable.

Sommaire

Retour sur ce que nous venons d'apprendre

Portée des variables

Considérons le code suivant :

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void echange1(int x,int y)
5  {
6      int tmp;
7      tmp=x;
8      x=y;
9      y=tmp;
10 }
11
12 int main()
13 {
14     int a=1,b=5;
15     printf("%d    %d\n",a,b);
16     echange1(a,b);
17     printf("%d    %d\n",a,b);
18     return 0;
19 }
```

Quel est l'affichage produit ?

Retour sur ce que nous venons d'apprendre

Portée des variables

Considérons le code suivant :

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  void echange2(int *x,int *y)
5  {
6      int tmp;
7      tmp=*x;
8      *x=*y;
9      *y=tmp;
10 }
11
12 int main()
13 {
14     int a=1,b=5;
15     printf("%d    %d\n",a,b);
16     echange2(&a,&b);
17     printf("%d    %d\n",a,b);
18     return 0;
19 }
```

Quel est l'affichage produit ?

Retour sur ce que nous venons d'apprendre

Portée des variables

Dessinez pour chacun des deux appels ce qui se passe dans la mémoire :

```c
void echange1(int x,int y)
{
    int tmp;
    tmp=x;
    x=y;
    y=tmp;
}
```

```c
void echange2(int *x,int *y)
{
    int tmp;
    tmp=*x;
    *x=*y;
    *y=tmp;
}
```