Les tableaux de caractères

Ils se traitent comme les autres tableaux (tri, recherches).
« On va avoir une différence de niveau cognitif entre l'humain et la machine ».

Exemple : Saisir du texte. => Suite de caractère => Donc pour la machine, ce n'est qu'un tableau.
L'algorithme que l'on a vu précédemment pour la saisie est inutilisable.
On va donc voir un ensemble de commandes spécifiques aux tableaux de caractères :

Pour faire une saisie de texte :
```
caractere texte [2000];
lire (texte); //global
```

0	1	2	3	4	5	6	…	1999
'B'	'O'	'N'	'J'	'O'	'U'	'R'	EOT	

EOT : End of text, le caractère utilisé pour délimiter la fin de la partie utile (caractère système)

➔ **Afficher un tableau de caractères**

1. Une boucle : tant que
2. Commande : caractere message[30000];
   lire (message);
   écrire (message); //affiche tous les éléments jusqu'à EOT (qui n'apparait pas).

En langage C/C++ : 2 possibilités

1. Avec cin >> / cout <<
   ```
   char txt[3000];
   cin >> txt; // = lire
   cout << txt; // = Ecrire
   ```
   Défaut : La commande peut s'arrêter aux espaces.

2. Il faut utiliser #include <cstring>
   Lire : gets
   Ecrire : puts

   Exemple :
   ```
   char tabc[10000];
   gets (tabc); // = lire
   puts (tabc); // = écrire
   ```

➔ **Taille utile d'un tableau de caractères**

Commande longueur-texte (en C : strlen) : donne le nombre de caractère avant le caractère EOT
Exemple :
caractère montexte[200] ;
entier taille;
lire (montexte);
taille <- longueur-texte (montexte);

➔ **Initialiser un tableau de caractères**

caractère tablo[1000];
tablo <- « salut » // Ce cas va être refusé

Deux solutions :

1. Possible à la définition du tableau
   caractère tablo[1000] <- « salut »;

2. Utiliser une commande de copie
   copier (en C : strcpy) = copier (destination, source);

   Copie tous les caractères de **source** et les recopie dans le tableau **destination** :
   copier (tablo, « salut »);

Donc tablo :

1	2	3	4	5	6	?	….
‘s’	‘a’	‘l’	‘u’	‘t’	EOT	?	…

➔ **Comparer 2 textes : dans l’ordre.**

Lexicographique (du dictionnaire).
Pas avec =, <, >
On compare les « cases » dans les tableaux

Commande : compar (en C : strcmp)
compar (tableau1, tableau2); // ou texte entre « »

Ça donne 0 si les deux textes stockés sont les mêmes (partie utile).

```
Caractere nom[50];
afficher («quel est votre nom :»);
lire(nom);
Si (compar(nom, «dupond»)=0)
{
    afficher («Vous êtes reconnu»);
}
Sinon
{
    afficher(«Accès refusé»);
}
```

/!\ NE PAS UTILISER AVEC DES TABLEAUX D'ENTIERS / DE REELS !

```
Entier tab[5];
tab[0] <- 39;
// On en initialise d'autres … On ne sait plus combien.
// Oups, on a besoin de la taille utile.
longueur-texte (tab) ; //???
```

➔ **Tableaux à plusieurs dimensions**

Exemple :

```
Réel tab [4][5][6][3][11][5]; // Ok, mais pas pratique à interpréter/visualiser en 2D
```

En pratique : Un jeu de plateau (échecs)
Echiquier = Plateau de 8x8 cases (64 cases)
Comment représenter l'état d'une case ?
1 : Pion blanc
0 : Vide

Donc l'état du jeu ⇔ 64 entiers ⇔ 1 tableau.

```
Entier échiquier [64];
```

Soucis : Comment traduire une position d'échec e5, f8, a3 en un indice entre 0 et 63.

<u>Utiliser un tableau à 2 dimensions (avec 2 indices)</u>.

```
entier échiquier[8][8]; // Soit 64 valeurs
```

Premier 8 : Lignes / Deuxième 8 : Colonnes

Pour repérer une valeur, il y a 2 indices : Le numéro de ligne et le numéro de colonne à prendre en compte.

```
échiquier[1][2];
échiquier[2][7];
échiquier[7][1];
```

Programme pour lister toutes les cases de l'échiquier sous forme d'entier :

```
entier echiquier[8][8];
entier lig, col;
// On remplit le tableau 2D
Pour lig de 0 à 7
{
   Pour col de 0 à 7
   {
      Afficher (echiquier [lig][col]),
   }
}
```

07/10/2013

Chapitre IV : Les pointeurs

Une variable :

- Un nom unique ⇔ Un endroit dans la RAM (mémoire vive : Numérique = Un nombre *)
- Un type
- Une valeur numérique (stockée en binaire)

* Ce nombre est une adresse (mémoire) noté @ (par abréviation).
= Un numéro de cellule mémoire, similaire à un indice, dans la RAM.

RAM

0
1
…
…
4 milliards

➔ **Notation informatique : &, se lit « adresse de ».**

Cette notation est toujours suivie d'une variable, et pas d'une expression.

Exemple (qui est en fait est un contre-exemple, oui c'est logique) :
&2 ← Adresse de la valeur 2, ça n'a pas de sens.

Entier a; ⇔ Une adresse lui est forcément associée lorsqu'elle est définie. Cette boîte est forcément quelque part.
a <- 5; ⇔ Ecrire la valeur 5 dans la cellule n°XXXX (décidé par le processeur).

Afficher(a); // 5. La valeur de la variable a est affichée, soit 5.
Afficher (&a); // ⇔ Afficher l'adresse de la variable a, soit XXXX (adresse de la cellule mémoire dans laquelle est stockée la variable a.

L'adresse est une information en lecture seule. On ne peut pas modifier l'@ d'une variable.

➔ **Qu'est-ce qu'un pointeur ?**

Un pointeur est une variable dont la valeur n'est ni un entier, ni un réel, ni un caractère, mais une adresse (= numéro de cellule mémoire, compris entre 0 et 2^{32}, soit 4 milliards environ).

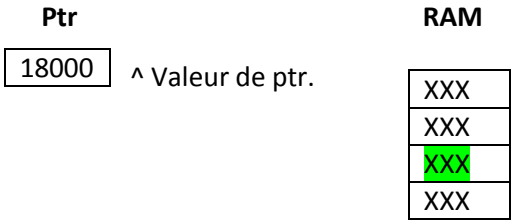
Une adresse prend des valeurs entières, mais ne se manipule pas comme une valeur de type entier.
➔ **C'est donc un nouveau type**.

Quel est le nom de ce type, comme char par exemple ? Eh bien, il n'y a pas de nom de fixe.

Un pointeur est une variable :

- Un nom
- Un type (que l'on ne sait pas nommer, mais qui existe)
- Une valeur
- Un contenu (totalement différent de la valeur) : C'est ce qu'il y a dans la mémoire, à l'adresse stockée dans le pointeur.

Supposons que : on sait définit un pointeur nommé ptr, et que sa valeur est par exemple 18'000.



XXX : Case numéro 18'000 = Le contenu de PTR.

La valeur est à gauche de la →
Le contenu est à droite de la →

Notation : * = Contenu de …

```
*ptr <- 10 ; // Contenu de ptr
afficher (ptr); // 18000 !
```

Son type :
ptr
18000 => [ ] Cellule n°18000
^ Type du contenu
= Une adresse

Est-ce que c'est un type réel ?
Alors, ça occupe 8 cellules : de 18000 à 18007

Si c'est de type caractère : 1 seule cellule (18000)

Pour définir et manipuler un pointeur

1. Une adresse
2. Le type du contenu (sert à définir du contenu)

Exemple : Soit pent un pointeur dont le contenu est de type entier.
On écrira : le contenu de pent est de type entier.

Entier *pent; // Le contenu de pent est de type entier.

Confusion de notation : Le contenu de pent est de type entier.
→ Donc, pent est un pointeur (qui pointe une valeur de type entier)

Entier *pent ; ⇔ *pent (=pent) * (=est un pointeur) entier (=de type entier).

Etoile (*) ne veut PAS DIRE POINTEUR !

➔ **Valeur initiale d'un pointeur**

Caractere *ptrc ;

/!\ Ne pas manipuler le contenu.
Sinon : CARTON ROUGE ! ET OUAIS QU'EST-CE TU CROIS, TOI ?!

Rappels :

& : « adresse de »
* : « contenu de »

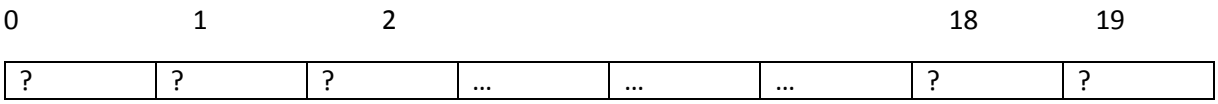
Un pointeur se définit par le type pointé :

- Entier *ptr ;
- Reel *ptrl ;
- Caractere *ptcar ;

Retour sur les tableaux

Ex :

entier tab[20]; // 80 octets



Stockées de manière contigue

Reel tab2[1000000] // 8000000 octets = 28 MO
Un tableau est un pointeur constant

Illustration :

Réel tabr[100]

Tabr <- qqchose = Impossible, car le tableau est constant.
Les tableaux définis avec des [ ] sont dits statiques.
Il en existe d'autres, les tableaux dynamiques.

Tableau statique : Taille maximale fixée à l'écriture du programme.
Tableau dynamique : Taille maximale donnée quand le programme s'exécute.

Principe : Demander de l'espace mémoire pour y stocker des valeurs. Avec les tableaux statiques, la demande traitée automatiquement
Cette allocation est faite automatiquement avec les tableaux dynamiques : demande d'allocation avec réservation

- ➔ On donne le type et la taille maximale (comme pour un tableau statique).
  La taille maximale peut être une variable.

La réservation répond par une adresse = Là où débute un bloc (zone) de mémoire suffisamment grande.

Illustration : Création d'un tableau dynamique d'entiers.

```
1 ligne [ entier *ptr ;
entier taille ;
faire
{
    affiche (« Entrez la taille de votre tableau : ») ;
    saisir(taille) ;
} tant que (taille <1)
```

Reservertation (taille entier) ; // Calcul de la taille de la zone

2) Adresse de cette zone fournie par la réponse :
Ex : La réponse vaut 3000

Principe des allocations dynamiques :

Reservation (50 caracteres) ← Demande de 50 octers dans la zone 1
ptrc ← Reservation (50 caracteres) ← Demande de 50 octets en zone 2 (obligatoire)

Ram

ZONE 1 POUR LE PROGRAMME (Occupée)

En C / C++ : Voir Poly

Allocation dynamique (suite)

Rappels :
Tableau statique : Allocation faite automatiquement

Exemple :

Caractère txt[30] ;

Txt [@] →

0	1	2	…	28	29

Vérrouillé — Mémoire allouée pour les éléments

Tableau dynamique :
But : Arriver à la même structure en mémoire en 2 étapes

1. **Définir un pointeur :**
   Exemple : caractère *txt ;

2. **Allouer de la mémoire pour les éléments**
   txt ← Réservartion(30 caractere) ;
   (Dévérouiller)

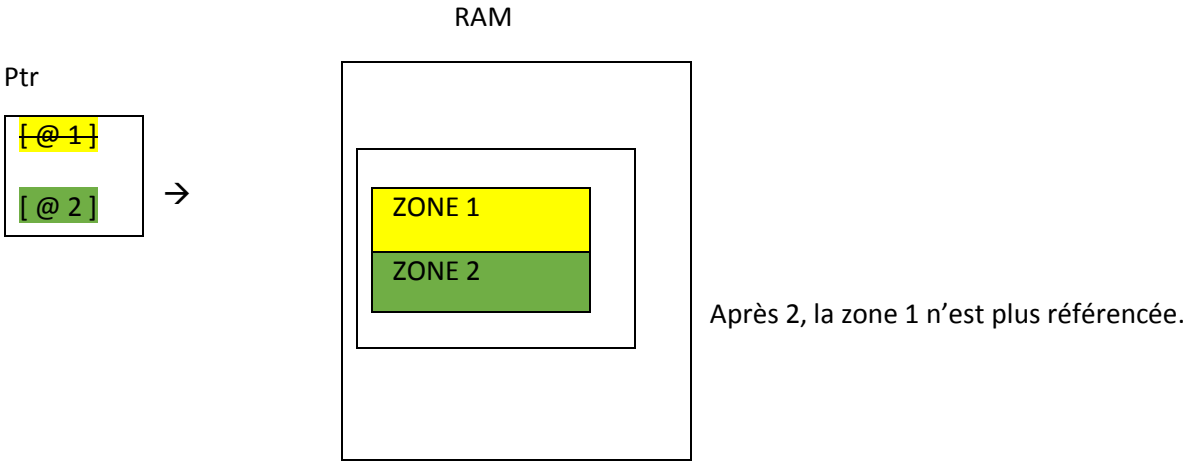
/!\ Aux zones de mémoires non pointées (non référencées)

Exemple :

```
Reel *ptr; //Pour un tableau dynamique
entier taille;
faire
{
    afficher (« taille du tableau ? »);
    saisir (taille);
} tant que (taille < 1)
ptr ← réservation (taille réel);

// On travaille avec le tableau
// Et puis, pouf, on en veut un d'une case en plus.

ptr ← réservation (taille+1 reel)
```



Après 2, la zone 1 n'est plus référencée.

Il faudrait donc pouvoir libérer une zone dont on n'a plus besoin pour pouvoir réutiliser la RAM.
Pour cela, il existe cela une commande pour libérer (donc l'inverse de la réservation) de la RAM pour s'en servir pour d'autres allocations futures.

Syntaxe de libérer :

Libérer (pointeur)
\ une adresse /

Effet : Libérer la zone pointée (ne détruit pas le pointeur)

En C / C++ : delete[] pointeur;

Les effets de new / delete :

```
Int main ()
{
   long * tablo;
   long size;
   do
   {
         cout << “Entrez la taille :”;
         cin >> size;
    } while (size < 1);
    tablo = new long [size];
     // New = reservation, long = type de chaque case, size = Nombre de case = taille max
     delete[] tablo;
     tablo = new long [size + 1]
```

Et si l’allocation n’est pas possible ? OH NO !

Exemple :

Entier *monptr;
monptr ← réservation (2'000'000'000 entier);

Peut-on allouer 8 GO ??

Si oui, on obtient une adresse.
Si non… On obtient quand même une adresse. Mais une adresse spéciale dont on sait qu’elle n’est pas valide. L’adresse ø : NULL

NULL : écriture qui signifie ø

```
Entier *monptr;
montptr ← réservation (2'000'000'000 entier);
si (monptr == NULL)
{
   afficher (« allocation impossible »);
}
sinon
{
    // Suite du programme
}
```

Utiliser la valeur NULL pour ‘’protéger’’ un programme.

Source de plantage : Erreur d’accès à une zone non allouée = Le pointeur désigne une adresse non autorisée. Il y a différents cas de figure :

1. Allocation impossible (parade connue juste au-dessus).
2. Pointeur non initialisé : Juste après sa définition.
3. Pointeur qui a été libéré (**delete[] / libérer()**)

Exemple :

```
reel *tabreel;
tabreel ← NULL;  // « Je sais que mon pointeur ne désigne pas une zone valide ».
// Beaucoup d’instructions
// …
// …
// Puis, on veut utiliser le tableau tabreel

si (tablreel == NULL)
{
   afficher(‘’attention : pointage vers null’’);
}

libérer (monptr);
// Immédiatement après (puisqu’il ne pointe plus vers une adresse valide, il faut faire ceci :
monptr ← NULL; // Il faut l’ajouter, car c’est pas fait automatiquement.
```