TP Programmation C/C++

Les chaînes de caractères et les tableaux

Éléments de cours

© 2017 *tv* <tvaira@free.fr> v.1.1

Sommaire

Les types dérivés **2**

Les chaînes de caractères **3**

Déclarations de chaînes de caractères . . . 3

Opérations sur les chaînes de caractères . . . 4

Lecture de chaînes de caractères . . . 4

Affichage de chaînes de caractères . . . 5

Les tableaux **6**

Déclarations de tableaux . . . 6

Les tableaux à plusieurs dimensions . . . 7

Parcourir un tableau . . . 8

Les tableaux et les pointeurs . . . 8

Trier un tableau . . . 9

Les tableaux dynamiques . . . 11

Conclusion **12**

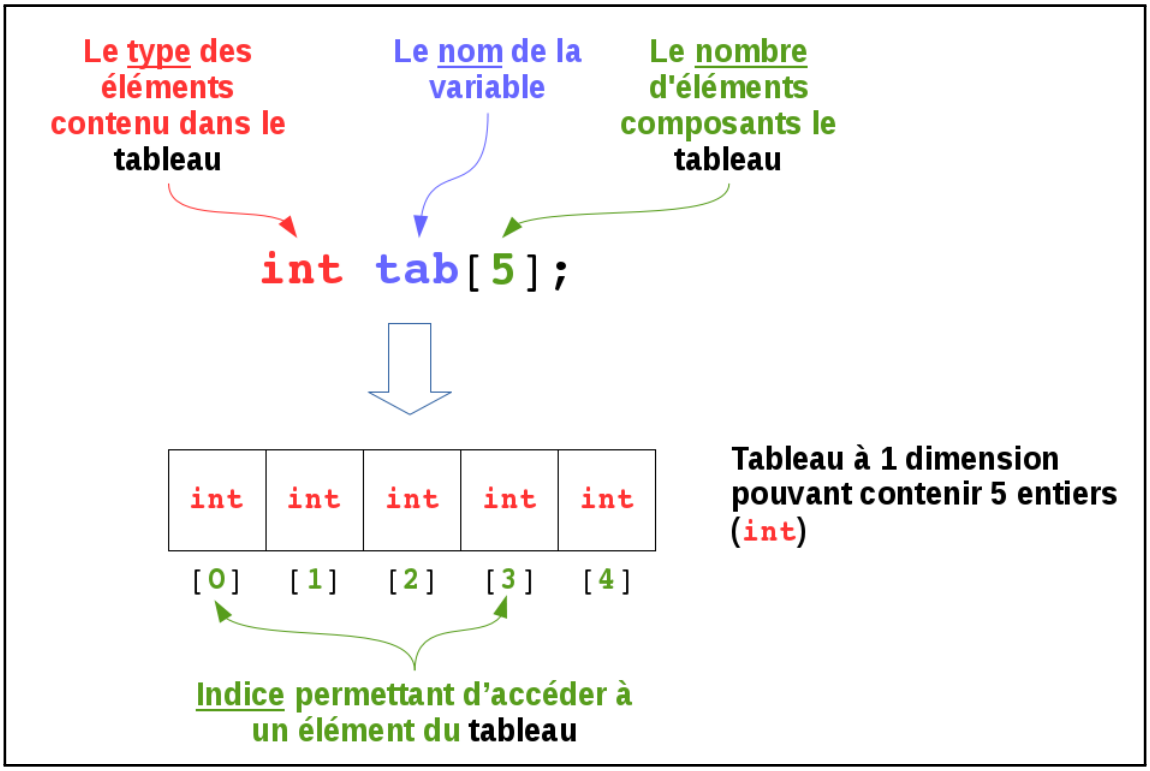
Les objectifs de ce tp sont de comprendre et mettre en oeuvre des variables de type tableau ainsi que la manipulation des chaînes de caractères.

Les types dérivés

Dans de très nombreuses situations, les types de base s'avèrent insuffisants pour permettre de traiter un problème : il peut être nécessaire, par exemple, de stocker un certain nombre de valeurs en mémoire afin que des traitements similaires leurs soient appliqués.

Dans ce cas, il est impensable d'avoir recours à de simples variables car tout traitement itératif serait inapplicable. D'autre part, il s'avère intéressant de pouvoir regrouper ensemble plusieurs variables afin de les manipuler comme un tout.

Pour répondre à tous ces besoins, le langage C/C++ comporte la notion de **type agrégé** en utilisant : les **tableaux**. Il existe aussi les **structures**, les **unions** et les **énumérations**.



Notion de tableau

Les tableaux sont des **conteneurs** (*container*), c'est-à-dire est un **objet qui contient d'autres objets**.

☞ *En C++, il existe de nombreux autres conteneurs (`vector`, `list`, `map`, `stack`, `queue`, ...). Un conteneur fournit généralement un moyen de gérer les objets contenus (au minimum ajout, suppression, parfois insertion, tri, recherche, ...) ainsi qu'un accès à ces objets.*

 © 2017 <tvaira@free.fr>

Les chaînes de caractères

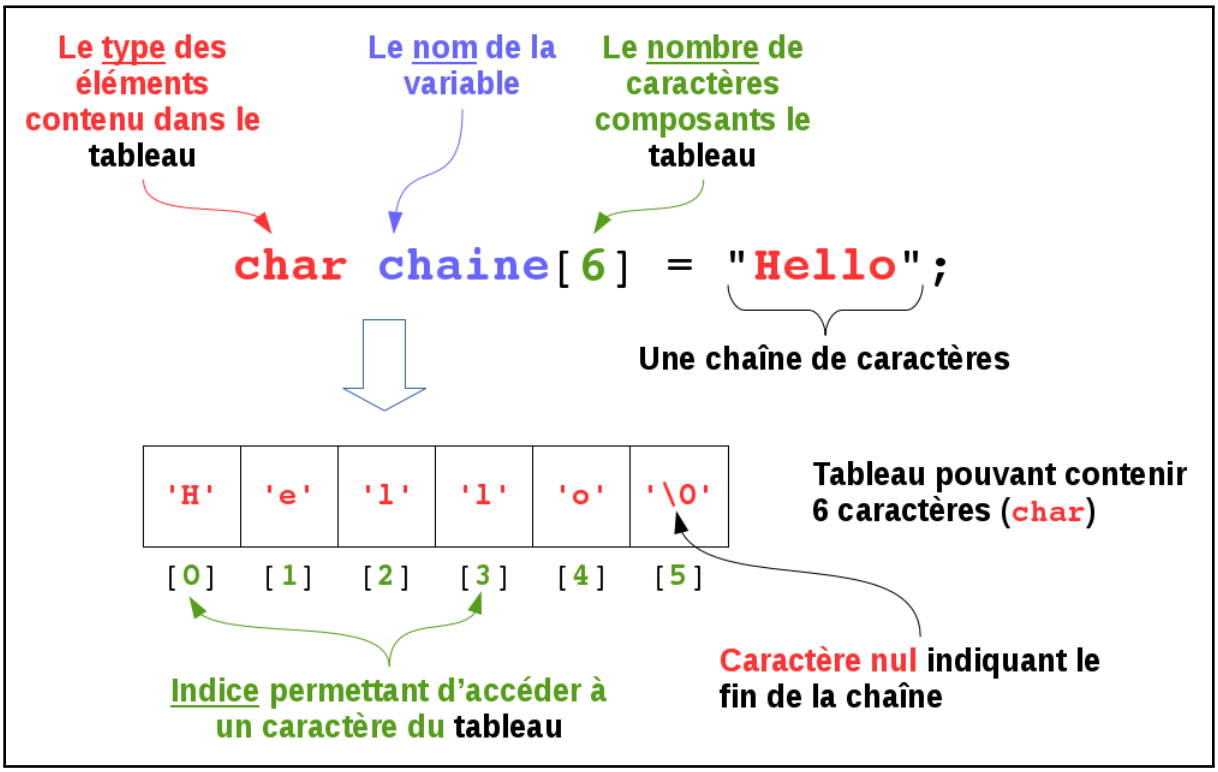
En C/C++, les **chaînes de caractères** sont délimitées par des guillements anglais ("). `"Hello world !\n"` est donc une chaîne de caractères. Le code \n est un "caractère spécial" indiquant le passage à une nouvelle ligne (*newline*).

Les **chaînes de caractères** contiennent donc des caractères ! En C/C++, ils sont codés par défaut en ASCII (sur 8 bits soit un octet). Un **caractère** est délimité par des guillements simples (’). En C/C++, un **caractères** est stocké dans une variable de type `char`.

☞ *man ascii*

Une **chaîne de caractères** est délimitée par un caracère spécial de **fin de chaîne** : le caractère nul qui a pour valeur `0` ou `’\0’`. Il est automatiquement ajouté lorsqu'on utilise la notation ("").

En C++, les **chaînes de caractères** sont stockées dans des variables de type `string`. En C, il n'y a pas de type dédié et on utilisera des tableaux de caratères.



Déclarations de chaînes de caractères

En C++, les **chaînes de caractères** sont stockées dans des variables de type `string` :

```
string prenom = "Robert"; // string pour les chaînes de caractères
```

Les chaînes de caractères en C++

En C, il n'y a pas de type dédié et on utilisera des tableaux de caratères :

```
// une chaîne de caractères délimitée par un fin de chaîne (le fin de chaîne est ajouté
   automatiquement ici)
char nom[] = "Robert"; // la taille est calculée automatiquement (ici 6+1 = 7 carctères)

// un autre tableau de 6 caractères peut stocker une chaîne de 5 caractères (6-1 pour le fin
    de chaîne)
```

 © 2017 <tvaira@free.fr>

```c
char autre[6];

// un tableau de caractères (ce n'est pas à proprement parler une chaîne de caractères car
   il n'y a pas de fin de chaîne)
char tab[4] = { 'a', 'b', 'c', 'd' };

// une chaîne de caractères (il y a le fin de chaîne)
char chaine[4] = { 'a', 'b', 'c', '\0' };

// Un simple caractère
char lettre = 'A'; // le caractère ASCII 'A' soit 0x41
```

Les chaînes de caractères en C

Opérations sur les chaînes de caractères

En C++, le type `string` permet d'utiliser les opérateurs courants : `=` pour l'affectation, `+` pour la concaténation ou `==` pour la comparaison par exemple. En C, ce n'est pas possible et il faut passer par des fonctions : `strcpy()` pour copier une chaîne, `strcat()` pour la concaténation ou `strcmp()` pour la comparaison. Ces fonctions ont besoin impérativement que les chaînes de caractères se terminent par un fin de chaîne (caractère nul). Attention aussi aux dépassements de taille !

☞ *man string*

```c
string nom = "Arizona";
string fils = nom + " Junior"; // Ajoute (concatène) des caractères à la fin

if (nom == fils) cout << "Les deux chaînes sont identiques.\n";
```

Opérations sur les chaînes de caractères en C++

☞ *http://www.cplusplus.com/reference/string/string/*

```c
#include <string.h> /* pour les fonctions str... */

char nom[] = "Arizona"; // ici autorisé car c'est une initialisation et non une affectation
char fils[16];

strcpy(fils, nom); // copie nom dans fils
strcat(fils, " Junior"); // concatène deux chaînes

// comparaison de chaînes
if (strcmp(nom,fils) == 0) printf("Les deux chaînes sont identiques.\n");
```

Opérations sur les chaînes de caractères en C

Lecture de chaînes de caractères

La lecture des chaînes de caractère (`scanf()` en C/C++ ou `cin` en C++) se termine sur ce qu'on appelle un espace blanc (*whitespace*), c'est-à-dire le caractère espace, une tabulation ou un caractère de retour à la ligne (généralement la touche Enter). Notez que les espaces sont ignorés par défaut.

```c
// En C/C++ :
char msg[16];
```

 © 2017 <tvaira@free.fr>

```
scanf("%s", &msg[0]); // %s pour une chaîne mais attention au dépassement de taille (on peut
    uliser fgets)
scanf("%15s", &msg[0]); // on peut limiter le nombre de caractères saisis (ici 15+1 = 16)

// si votre saisie comporte des espaces, vous pouvez utiliser :
// une expression rationnelle (entre crochets) acceptant tous les caractères sauf (^) le
   retour à la ligne (\n) :
scanf("%15[^\n]s", &msg[0]);

// ou la fonction fgets
fgets(&msg[0], 16, stdin); // attention fgets stocke le retour à ligne

// En C++ :
string message; // chaîne de caractères seulement en C++
cin >> message;

// si votre saisie comporte des espaces, il vous faudra alors utilisé la fonction getline
// cf. http://www.cplusplus.com/reference/string/string/getline/
getline(std::cin, message);
```

Lecture de chaînes de caractères en C/C++

✍ *Sous Linux, vous pouvez indiquer la fin d'un flux en combinant les touches* Ctrl + d *qui indiquera qu'il n'y a plus de saisie. Vous pouvez aussi stopper le programme avec* Ctrl + z *ou l'interrompre avec* Ctrl + c .

Les caractères tapés ne sont pas directement transmis au programme, mais placés (par le système) dans un tampon (*buffer*). Il est possible qu'il reste des caractères d'une saisie précèdente et donc vous aurez besoin de vider le *buffer* avant de (re)faire une saisie. La méthode pour vider le buffer clavier (*stdin*) consiste à consommer tous les caractères présents dans ce buffer jusqu'à ce qu'il soit vide :

```
// En C/C++ :
int c = 0;
while ((c = getchar()) != '\n' && c != EOF);

// Ou :
scanf("%*[^\n]");
getchar();

// En C++ :
cin.clear();
cin.ignore(numeric_limits<streamsize>::max(), '\n');
```

Vider le tampon stdin

✍ *EOF signifie End Of File. Pour générer un EOF avec le clavier, il suffit de taper, en début de ligne,* Ctrl + z *sous DOS/Windows et* Ctrl + d *sous UNIX, puis de valider par Entrée.*

Affichage de chaînes de caractères

Pour l'affichage des chaînes de caractère, on utilise `printf()` en C/C++ ou `cout` en C++ :

```
char msg[] = "Bonjour"; // chaîne de caractères C/C++
printf("msg : %s contient %d caractères\n", msg, strlen(msg)); // %s pour une chaîne se
   terminant par un fin de chaîne
```

 © 2017 <tvaira@free.fr>

```
printf("premier caractère : %c\n", msg[0]); // %c pour un simple caractère

string message = "Bonjour"; // chaîne de caractères seulement en C++
cout << "msg : " << msg << " contient " << msg.length() << " caractères" << endl; // endl
    est équivalent à \n
cout << "premier caractère : " << msg[0] << "\n";
```

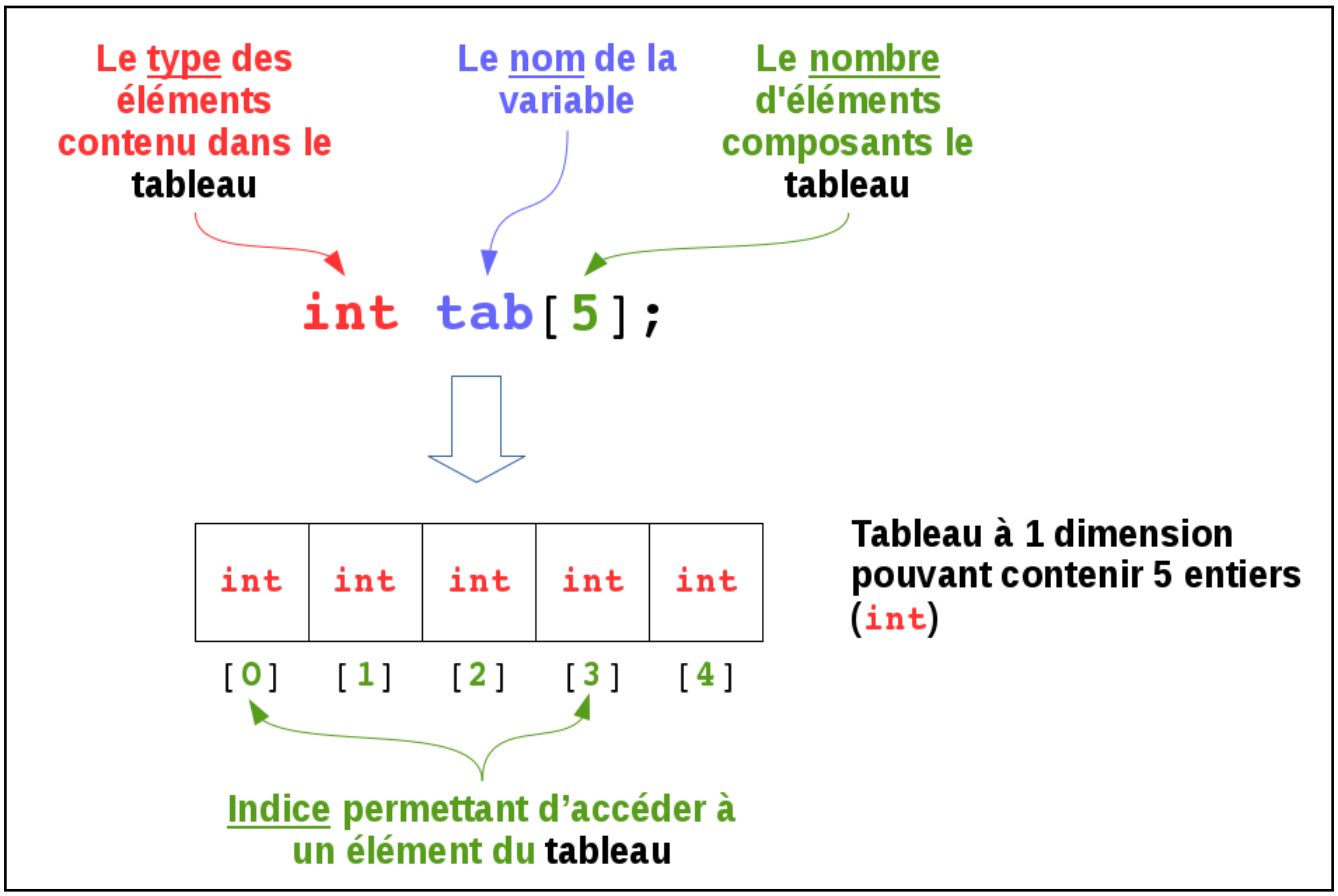
Affichage de chaînes de caractères en C/C++

✍ *L'affichage est bufferisé lui aussi. Tant que le tampon n'est pas plein, les caractères transmis ne seront pas effectivement affichés mais tout simplement placés dans le tampon. Pour vider le contenu du tampon vers l'écran, il faut un retour à la ligne (`\n`) ou que le tampon soit plein. On peut aussi forcer le vidage de ce tampon à l'aide de la fonction `fflush(stdout)` ou carrément ne pas utiliser de buffer en appelant `setbuf(stdout, NULL)`.*

Les tableaux

Un **tableau** est un **ensemble d'éléments de même type** désignés par un identificateur unique (un nom). Chaque élément est repéré par une valeur entière appelée **indice** (ou index) indiquant sa position dans l'ensemble.

Les tableaux sont toujours à bornes statiques et leur indiçage démarre toujours à partir de `0`.



Déclarations de tableaux

La forme habituelle de déclaration d'un tableau est la suivante :
`type identificateur`[$dimension_1$]`...` [$dimension_n$]

 © 2017 <tvaira@free.fr>

Exemple de déclarations de tableaux :

```
int notes[1000]; // un tableau de 1000 int non initialisé

float notes[1000]; // un tableau de 1000 float non initialisé
```

Déclarations de tableaux en C/C++

☞ *Par "non initialisé", on entend qu'il y a une valeur dans chaque case du tableau mais on ne la connaît pas. Il faut donc considérer qu'il y a "n'importe quoi" ! On rappelle que "rien" n'est pas une notion en informatique car les bits prennent soit la valeur `0` soit la valeur `1` : il n'y a pas de valeur "rien".*

Certains cas d'initialisation de tableaux sont admis :

```
int notes[1000] = {0}; // un tableau de 1000 int initialisé à 0

float f[1000] = {0.}; // un tableau de 1000 float initialisé à 0

int coefficients[4] = { '1', '2', '2', '4' }; // un tableau de 4 entiers

// La dimension d'un tableau peut être omise si le compilateur peut en définir la valeur
float t[] = {2., 7.5, 4.1}; // tableau de 3 éléments

// Et seulement avec le compilateur g++ :
// On peut utiliser une variable
int nbProduits = 1000;
int stock[nbProduits] = {0};
// et une saisie
int nbCases;
scanf("%d", &nbCases);
int cases[nbCases] = {0}; // certaines versions n'admettent pas l'initialisation
```

Déclaration et initialisation de tableaux en C/C++

Sinon il vous faudra le faire manuellement :

```
int nbEleves = 30;
int presences[nbEleves];
int i;

// Avec une boucle :
for(i=0;i<nbEleves;i++)
{
    presences[i] = 1; // ici 1 est la valeur initiale
}

// Avec la fonction memset :
memset(&presences[0], 0, 30*sizeof(int));
```

Initialisation de tableaux en C/C++

Les tableaux à plusieurs dimensions

✍ *Contrairement à beaucoup d'autres langages, il n'existe pas en C de véritable notion de tableaux multidimentionnels. De tels tableaux se définissent par composition de tableaux, c'est à dire que les éléments sont eux-mêmes des tableaux.*

 © 2017 <tvaira@free.fr>

```
// tableau à 2 dimensions de 2 lignes et 5 colonnes :
int  m[2][5] = { 2, 6, -4, 8, 11,  // initialise avec des valeurs
                 3, -1, 0, 9, 2 };

// tableau à 2 dimensions pour stocker plusieurs chaînes de caractères
char noms[][16] = { {"robert"},
                    {"roger"},
                    {"raymond"},
                    {"alphonse"} };

int  x[5][12][7]; // tableau a 3 dimensions, rarement au-delà de cette dimension
```

Les tableaux à plusieurs dimensions en C

Parcourir un tableau

Pour parcourir un tableau, il faut utiliser une boucle :

```
int nbQuestions = 20;
int points[nbQuestions];
int i;

// saisir des données dans un tableau :
for(i=0;i<nbQuestions;i++)
{
    scanf("%d", &points[i]);
}

// afficher des données d'un tableau :
for(i=0;i<nbQuestions;i++)
{
    printf("%d\n", points[i]);
}
```

Parcourir un tableau en C/C++

☮ Le plus grand danger dans la manipulation des tableaux est d'accéder en écriture en dehors du tableau. Cela provoque un accès mémoire interdit qui n'est pas contrôlé au moment de la compilation. Par contre, lors de l'exécution, cela provoquera une exception de violation mémoire (*segmentation fault*) qui se traduit généralement par une sortie prématurée du programme avec un message "Erreur de segmentation".

Les tableaux et les pointeurs

✍ *L'identificateur du tableau (le nom de la variable) ne désigne pas le tableau dans son ensemble, mais plus précisément l'adresse en mémoire du début du tableau (**l'adresse de la première case**). Ceci implique qu'il est impossible d'affecter un tableau à un autre. L'identificateur d'un tableau sera donc "vu" comme un **pointeur constant**.*

```
#define MAX 20  // définit l'étiquette MAX egale à 20

// Attention :
int a[MAX], b[MAX];
```

 © 2017 <tvaira@free.fr>

```
a = b; // cette affectation est interdite ! Il faudra faire une boucle pour traiter chaque
    case
```

La dimension d'un tableau peut être omise dans 2 cas :

– le compilateur peut en définir la valeur

```
int t[] = {2, 7, 4}; // tableau de 3 éléments
```

– l'emplacement mémoire correspondant a été réservé

```
// la fonction fct admet en parametre
void fct(int t[]) // un tableau d'entiers qui existe déjà
```

✍ *Lorsque le nom d'un tableau constitue l'argument d'une fonction, c'est l'adresse du premier élément qui est transmise. Ses élément ne sont donc pas recopiés. Lorsque l'on passe un tableau en paramètre d'une fonction, il n'est pas possible de connaître sa taille et il faudra donc lui passer aussi sa taille.*

Utilisation des pointeurs avec les tableaux :

```
int t[5] = {0, 2, 3, 6, 8}; // un tableau de 5 entiers
int *p1 = NULL; // le pointeur est initialisé à NULL (précaution obligatoire)
int *p2; // pointeur non initialisé : il pointe donc sur n'importe quoi (gros danger)

p1 = t;             // p1 pointe sur t c'est-a-dire la première case du tableau
// identique a : p1 = &t[0];

p2 = &t[1];         // p2 pointe sur le 2eme élément du tableau

*p1 = 4;             // la première case du tableau est modifiée
printf("%d ou %d\n", *p1, t[0]); // affiche 4 ou 4

printf("%d ou %d\n", *p2, t[1]); // affiche 2 ou 2
p2 += 2;             // p2 pointe sur le 4eme élément du tableau (indice 3)
printf("%d ou %d\n", *p2, t[3]); // affiche 6 ou 6

// on peut utiliser les [] sur un pointeur :
p1[1] = 8;           // identique à : *(p1+1) = 8; ou a : t1[1] = 8;
printf("%d\n", t[1]); // affiche 8

// et inversement :
*(t1+1) = 10;        // identique à : t1[1] = 10;
printf("%d\n", t[1]); // affiche 10
```

Les tableaux et les pointeurs

Trier un tableau

La bibliothèque C standard offre une fonction de tri : `qsort()`. Elle sert à trier un tableau d'éléments à l'aide d'une fonction de comparaison à fournir. Pour l'utiliser, vous devez inclure le fichier `<stdlib.h>`.

`qsort()` trie les éléments dans l'ordre que vous lui demandez. Plus précisément, vous donnez à `qsort()` une fonction de comparaison. Cette fonction prend deux éléments A et B, et indique si A doit être avant ou après B dans le tableau trié. La fonction de comparaison est donc vitale, c'est elle qui indique dans quel ordre et selon quels critères il faut trier le tableau.

La fonction de comparaison a le prototype suivant : `int compareValeurs(const void* val1, const void* val2)`.

 © 2017 <tvaira@free.fr>

Elle reçoit en argument deux pointeurs `val1` et `val2`, et retourne un entier. `val1` et `val2` sont des pointeurs constants sur les éléments à comparer. Le `const` empêchera la fonction de comparaison de modifier le contenu du tableau pendant le tri. Le type `void *` est ici obligatoire car la fonction ne peut pas connaître avant le type des éléments à comparer. Elle utilise donc des pointeurs génériques (`void *`) qu'il faudra "caster" (transtyper) vers les types désirés.

✍ *Si la fonction de comparaison est `<=`, alors à la fin du tri du tableau `a[0..n-1]` les éléments sont réordonnés de telle manière que `a[0] <= a[1] <= ... <= a[n-1]`. Si on utilise la fonction `>=`, cela donnera `a[0] >= a[1] >= ... >= a[n-1]`.*

Seul le signe de la valeur retournée par la fonction de comparaison compte. Par exemple on peut retourner `-1` pour A avant B, `+1` pour A après B, et `0` (zéro) si l'ordre ne compte pas.

```
int compareEntiers(const void* val1, const void* val2)
{
    int i1 = *(const int*)val1; // on caste sur un type entier
    int i2 = *(const int*)val2; // on caste sur un type entier

    if (i1 == i2)
        return 0;
    if (i1 < i2)
        return -1;
    else
        return +1;
}
```

Fonction de comparaison de deux entiers

En pratique, pour trier un tableau `valeurs` contenant `nbValeurs` éléments, on donnera à la fonction `qsort()` les arguments suivants :

- Le nom du tableau, `valeurs`
- Le nombre d'éléments du tableau, `nbValeurs`
- La taille en octets de chaque élément, `sizeof(valeurs[0])`
- Le nom de la fonction de comparaison, `compareEntiers`

```
#include <stdlib.h>

int compareEntiers(const void* val1, const void* val2)
{
    int i1 = *(const int*)val1;
    int i2 = *(const int*)val2;
    if (i1 == i2)
        return 0;
    if (i1 < i2)
        return -1;
    else
        return +1;
}

qsort(valeurs, nbValeurs, sizeof(valeurs[0]), compareEntiers);
```

Tri d'un tableau en C

✍ *Pour rechercher des valeurs dans un tableau, vous pouvez utiliser la fonction `bsearch()` de la bibliothèque C standard, déclarée dans le fichier `<stdlib.h>`. Cette fonction, qui ressemble beaucoup à `qsort()`, permet d'effectuer une recherche dichotomique. `bsearch()` recherche une valeur dans un*

 © 2017 <tvaira@free.fr>

tableau initialement trié et prend en argument l'objet cherché (la clé), le tableau, et la fonction de comparaison qui permet de savoir si un élément donné est situé avant ou après la clé, ou bien est égal à la clé.

☞ *En C++, il existe aussi une fonction* `sort()` *pour le tri de conteneurs.*

Les tableaux dynamiques

Pour allouer dynamiquement des tableaux en C, il faut utiliser les fonctions `malloc()` et `free()` :

```
int *T; // pointeur sur un entier

// allocation dynamique d'un tableau de 10 int (un int est codé sur 4 octets)
T = (int *)malloc(sizeof(int) * 10); // alloue 4 * 10 octets en mémoire

// initialise le tableau avec des 0
for(int i=0;i<10;i++)
{
   *(T+i) = 0;   // les 2 écritures sont possibles
   T[i]   = 0;   // identique à la ligne précèdente
}
// ou avec la fonction memset
memset(T, 0, sizeof(int)*10); // il faudra alors inclure string.h

// libération de la mémoire
free(T);
```

Allocation dynamique en C

✍ *La fonction* `realloc()` *modifie la taille d'un bloc de mémoire déjà alloué.*

Pour allouer dynamiquement des tableaux en C++, il faut utiliser les opérateurs `new` et `delete` :

```
#include <iostream>
#include <new>

using namespace std;

int main ()
{
   int *t = new int[5]; // alloue un tableau de 5 entiers en mémoire

   // initialise le tableau avec des 0
   for(int i=0;i<5;i++)
   {
      *(t + i) = 0;  // les 2 écritures sont possibles
      t[i]     = 0;   // identique à la ligne précèdente
      cout << "t[" << i << "] = " << t

   delete [] t; // libère la mémoire allouée

   return 0;
}
```

Allocation dynamique en C++

☞ *En C++, il existe aussi des tableaux dynamiques "prêts à l'emploi" avec le type* `vector`.

 © 2017 <tvaira@free.fr>

Conclusion

Conclusion : Les types sont au centre de la plupart des notions de programmes corrects, et certaines des techniques de construction de programmes les plus efficaces reposent sur la conception et l'utilisation des types.

Rob Pike (ancien chercheur des Laboratoires Bell et maintenant ingénieur chez Google) :

> « *Règle n°5 : Les données prévalent sur le code. Si vous avez conçu la structure des données appropriée et bien organisé le tout, les algorithmes viendront d'eux-mêmes. La structure des données est le coeur de la programmation, et non pas les algorithmes.* »

Cette règle n°5 est souvent résumée par « Écrivez du code stupide qui utilise des données futées ! ».

 © 2017 <tvaira@free.fr>