

Les tableaux

Il est possible en C de déclarer un tableau contenant des données de types identiques qui sont ensuite rangées en mémoire dans des cases contigues.

Un tableau en C est une adresse mémoire (appelée pointeur) donc quand on demande le premier élément, on prends la première case au niveau du pointeur. Pour prendre le deuxième, on va une case plus loin et ainsi de suite.

Ce qui signifie que l'on peut aller plus loin que la taille réservée du tableau, lorsque l'on fait ça on dit que l'on "jardine en mémoire" ce qui est risqué car cela peut faire planter le programme et que les données en dehors du tableau peuvent être réécrites par d'autres variables dans le programme.

Définition d'un tableau

```
/* Définition d'un tableau de 10 entiers simple */
int suite[10];

/* Initialisation d'un tableau de 5 entiers */
int suite[10] = { 1,2,3,4,5 };

/* Lecture du troisième élément d'un tableau */
printf("%d\n", suite[2]);

/* Ecriture d'un élément du tableau */
suite[2] = 42;
```

Attention à l'initialisation des variables et tableaux

Cela veut aussi dire que si un tableau n'est pas initialisé, si on récupère une position qui n'a pas encore été définie on peut tomber sur d'anciennes valeurs encore dans la mémoire.

C'est pour cela qu'il faut généralement garder une variable supplémentaire pour garder le compte du nombre de cases écrites du tableau (cela n'est pas nécessaire pour les chaînes de caractère car on sait que la chaîne se finit au caractère `\0`). La longueur des données réellement dans un tableau est appelée "taille effective" tandis que la taille en mémoire du tableau est appelée taille physique.

Pour ce qui est des variables c'est pareil, par défaut les variables ne sont pas initialisées (bien que cela peut varier des OS et des compilateurs). C'est pourquoi il vaut toujours mieux initialiser les variables. Car les variables sont simplement des adresses mémoires, donc si on lit une variable

non initialisée on va lire les données qui sont à cet endroit dans la mémoire (il peut donc y avoir un peu n'importe quoi).

Voici un exemple de code permettant de tester cela :

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

int main(void) {
    /* On crée une variable non initialisée */
    int ma_variable;

    /* On demande à sscanf d'initialiser la variable à partir de rien : spoiler il va pas
    l'initialiser */
    sscanf("", "%d", &ma_variable);

    /* On lit la variable non initialisée */
    printf("Ma variable non-initialisée vaut : %d\n", ma_variable);

    return EXIT_SUCCESS;
}
```

Tableaux et fonctions

On peut passer des tableaux en arguments de fonctions cependant étant donné que le tableau est un pointeur il ne sera pas copié car c'est bien sa référence qui sera passée à la fonction.

Une conséquence de ça c'est qu'une fonction en C ne peut pas retourner un tableau, pour traiter des tableaux il vaut mieux passer le tableau en argument, le modifier dans la fonction et retourner la taille effective du tableau sous forme de int.

Voici un exemple simple d'utilisation de tableaux dans des fonctions :

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

int add_42(int tableau[], int taille_tableau);

int main(void) {
    /* On crée un tableau avec une taille de 10 contenant 3 éléments */
    int tableau[10] = {1,2,3};

    /* On note la taille effective du tableau comme étant 3 (pour les 3 éléments) */
```

```
int taille_tableau = 3;

/* On passe le tableau et la taille dans la fonction qui va modifier le tableau et retourner
la nouvelle taille */
taille_tableau = add_42(tableau, taille_tableau);

/* On affiche le nouvel élément de notre tableau */
printf("Le nouvel élément du tableau est %d\n", tableau[taille_tableau - 1]);

/* On ferme le programme */
return EXIT_SUCCESS;
}

/* On définit une fonction retournant un entier (nouvelle taille), un tableau de taille
indéfinie, et la taille effective du tableau */
int add_42(int tableau[], int taille_tableau) {
    tableau[taille_tableau] = 42;
    return taille_tableau + 1;
}
```

Revision #1

Created 2 October 2023 18:32:30 by SnowCode

Updated 6 January 2024 19:14:47 by SnowCode