TD 4 — Pointeurs et tableaux

Le langage C propose manières de manipuler des données complexes permettant au programmeur de construire ses propores types de données à partir des types primitifs :

les tableaux qui peuvent contenir un certain nombre de données d'un même type ;

les structures qui peuvent contenir des données de types différents.

Ces deux types de données combinés aux pointeurs permettent de définir de nombreux types de données structurées. Dans cette ressource introductive, nous l'annons pas aborder les structures, mais seulement les tableaux.

Rappel sur les tableaux

On rappelle qu'en algorithmique, un *tableau* est une suite de variables toutes de même type, stockées de manière contiguë dans la mémoire vive.

Chaque élément d'un tableau est muni d'un indice qui permet d'accéder à sa valeur. Ainsi, si `tab[]` désigne un tableau, la notation `tab[i]` permet d'accéder à la variable en position `i` du tableau. Attention, la première valeur est d'indice `0`.

En langage C, les tableaux fonctionnent comme des pointeurs, notion que nous allons aborder dans la première partie de ce TD. La deuxième partie sera consacrée aux tableaux.

Travail à rendre

Vous devrez déposer sur EPREL une archive `zip` nommée `R101_TD4_<NOMDEFAMILLE>.zip`, contenant un projets `Code::Blocks`.

Ce projet contiendra les réponses aux exercices. Vous veillerez à ce que les affichages dans la console permettent de facilement se repérer à travers les exercices.

1 Pointeurs

En langage C, les variables sont *typées* et représentent un emplacement de la mémoire réservé pour notre programme. Par exemple, l'instruction `char lettre;` crée une variable de type `char` : lors de l'exécution du programme, le système d'exploitation va réserver une zone dans la mémoire pour que l'on y stocke les valeurs prises par `lettre` pendant l'exécution.

L'endroit où chaque variable est stockée est repéré par une *adresse* unique. On y accède via l'opérateur `&`. L'adresse de la variable `lettre` est donc `&lettre`. On peut stocker cette adresse dans une variable : `char* ptr_lettre = &lettre;` Cette instruction crée un pointeur vers une variable de tpe `char` et lui affecte comme valeur lune adresse.

Exercice 1. On considère l'extrait de code suivant :

```c
int a = 42;
int* p = &a;

printf("%p\n", p);
printf("%d\n", *p);
```

Expliquer ce que font les deux `printf`. En particulier, quel est l'effet de l'opérateur $*$ dans l'expression `*p` ?

Exercice 2. Écrire un programme qui échange les valeurs de deux variables `a` et `b` *en utilisant deux pointeurs* `double* p_a` et `double* p_b` qui pointent respectivement vers les variables `a` et `b`. Voici un exemple de résultat attendu :

```
Saisir la valeur de a : 3.14
Saisir la valeur de b : 2.72

Après échange, la valeur de a est 3.72
Après échange, la valeur de b est 3.14
```

2 Tableaux

Voici plusieurs manières de déclararer un tableau, en initialisant ou pas ses valeurs :

```c
int tab1[4]; // Déclaration d'un tableau non initialisé de taille 4.
int tab2[4] = {6, 4}; /* Déclaration d'un tableau de taille 4.
                         tab1[0] = 6, tab2[1] = 4. Les autres valeurs valent 0. */
int tab3[] = {6, 4, 8, 9, 1, 0}; // Création et initialisation d'un tableau de taille 6.
```

2.1 Manipulation de tableaux

Exercice 3. Compléter l'extrait de code source suivant, pour afficher à l'aide d'une boucle, toutes les valeurs du tableau avec une valeur par ligne. Quel type de boucle faut-il utiliser ?

```c
int main(void) {
    int tab[] =  {6,4,9,1,0,8};

    /* À compléter */
}
```

Exercice 4. Écrire un programme qui déclare le tableau `{6.2, 4.0, 9.7, -1.8, 0.7, 0.8}` et calcule la somme de ses éléments.

Exercice 5. Ecrire un programme qui détermine la plus grande valeur dans un tableau d'entiers. Afficher ensuite la valeur et la position de cette valeur dans le tableau. Si le tableau contient plusieurs maxima, le programme retiendra la position du premier maximum rencontré.

2.2 Lien entre tableux pointeurs

Lorsque l'on crée un tableau en C, cela revient à créer un pointeur vers le premier élément. Plus précisément, considérons par exemple le tableau `int tab[3]`. Alors, on a l'égalité `tab == &(tab[0])` : la valeur `tab` est l'adresse du premier élément.

Cette égalité, un peu étrange au premier abord, est en fait très commode à manipuler, car elle permet de bénéficier de l'arithmétique des pointeurs, voir l'exercice ci-dessous.

Exercice 6 (Arithmétique des pointeurs)**.** On considère l'extrait de code suivant :

```c
int tab1[] = {10, 11, 12, 13, 14};
char tab2[] = {'a', 'b', 'c', 'd'};

int* p_tab1 = tab1;
int* p_tab2 = tab2;
```

1) Que représentent les pointeurs `p_tab1` et `p_tab2` ? Que valent `*ptab_1` et `*p_tab2` ?
2) Calculer et afficher `p_tab2++` et `p_tab2+=2`. Comment fonctionnent les incrémentations et les additions avec les pointeurs ?

Exercice 7. Déclarer un tableau statique de taille 7 et intialiser ses valeurs. Parcourir le tableau et afficher ses valeurs en utilisant uniquement une boucle for et un pointeur vers les éléments du tableau. On n'utiisera pas la notation avec les crochets pour parcourir le tableau.

2.3 Tableaux dynamiques

En langage C, un tableau dynamique (également appelé tableau alloué dynamiquement ou tableau à mémoire dynamique) est un type de structure de données qui permet de stocker un ensemble d'éléments de manière flexible. Contrairement à un tableau statique, vu précédemment, la taille d'un tableau dynamique n'est pas déterminée à la compilation, mais elle peut être déterminée et modifiée au cours de l'exécution du programme.

Les tableaux dynamiques sont généralement implémentés à l'aide de pointeurs et de fonctions d'allocation de mémoire dynamique, telles que `malloc()` et `free()`.

Voici comment créer et utiliser un tableau dynamique en C :

Allocation de mémoire On utilise la fonction `malloc()` (*memory allocation*) pour allouer dynamiquement de la mémoire. Cette fonction reçoit en paramètre la taille, *en nombre d'octets* que l'on souhaite obtenir. Elle renvoie un pointeur vers l'adresse qui a été réservée en mémoire. Ainsi, pour allouer un tableau de 10 `int`, on écrit :

```c
int* tableauDynamique = (int*) malloc(10*sizeof(int));
```

L'opérateur `sizeof` renvoie le nombre d'octets occupés par un type de variables. L'opérateur `(int*)` qui précède l'appel à la fonction `malloc()` s'appelle le *cast* : cela permet de préciser le type des variables – ici, ce sont des `int` –. Cette opération est nécessaire car la fonction `malloc()` renvoie un pointeur dont le type doit être précisé.

Si l'allocation a échoué, la valeur de retour sera `NULL`. Il faut donc toujours vérifier, après l'allocation, que le pointeur retourné est valide (non `NULL`).

Utilisation du tableau On manipule un tableau dynamique de la même manière qu'un tableau statique.

Modification de la taille On peut changer la taille d'un tableau avec la fonction `realloc()`. Par exemple, le code suivant augmente la taille du tableau de 10 à 20 éléments :

```c
int* tableauDynamique = (int*) malloc(tableauDynamique, 20*sizeof(int));
```

Cette fonction conserve les données existantes et agrandit la mémoire allouée si nécessaire.

Libération de la mémoire Après avoir terminé l'utilisation d'un tableau dynamique, il faut libérer la mémoire utilisée au moyen de l'instruction `free(tableauDynamique);` : cela évite les fuites de mémoire.

Exercice 8. Proposer un programme qui permet d'allouer un tableau d'entiers et de l'initialiser avec des 0. Le nombre de valeurs contenues dans le tableau devra être saisie au clavier. Un affichage des valeurs du tableau doit permettre, à la fin, de vérifier le bon déroulement des opérations d'allocation et d'initialisation. On pensera à libérer convenablement la mémoire.

Exercice 9.

1) Reprendre l'exercice précédent et redimensionner le tableau en doublant sa taille initiale.

2) Penser à initialiser à 0 les nouveaux emplacements obtenus.

3) Afficher le contenu du tableau redimensionné.

Exercice 10. Écrire un programme qui effectue les opérations suivantes :

1) Demander à l'utilisateur·rice de saisir la taille initiale d'un tableau dynamique de flottants.

2) Allouer dynamiquement de la mémoire pour le tableau en fonction de la taille saisie.

3) Demander à l'utilisateur·rice de saisir les valeurs pour les éléments du tableau.

4) Afficher les valeurs du tableau avec une précision de 2 chiffres après la virgule.

5) Permettre à l'utilisateur·rice d'ajouter, ou non, un élément supplémentaire au tableau.

6) Afficher à nouveau le tableau avec l'élément ajouté.

7) Libérer la mémoire allouée dynamiquement avant que le programme ne se termine.