Algorithmique et structures de données

Loïc Demange
loic.demange@etud.univ-paris8.fr

05 novembre 2021

TP précédent

Objectifs du TP :

- Coder la fonction **change** pour que le contenu de la variable *a* soit modifié en 4.
- Créer un tableau et afficher l'adresse de la première et de la deuxième case. Constater pourquoi le type est important.
- Créer un tableau d'entiers de 10 cases et le remplir, puis afficher son contenu.
- Créer une fonction qui prend en paramètre un tableau d'entiers et sa taille, et qui affiche son contenu.

TP précédent

Coder la fonction change pour que le contenu de la variable *a* soit modifié en 4.

```
#include <stdio.h>
void change(int *p)
{
    (*p)++;
}
int main()
{
    int a = 3;
    change(&a);
    printf("%d\n", a); // doit afficher 4
    return 0;
}
```

On est obligé de manipuler des adresses, sinon la variable sera copiée localement dans la fonction **change** et la variable initiale ne sera donc pas modifiée.

Ici, le pointeur est copié localement, mais ça pointe toujours sur l'adresse de *a*.

TP précédent

Créer un tableau et afficher l'adresse de la première et de la deuxième case. Constater pourquoi le type est important.

```
#include <stdio.h>
int main()
{
    int tab[2] = {1,2};
    printf("1er case:%p\n2eme case:%p\n", tab, &tab[1]);

}
```

On peut constater qu'il n'y a que 4 (ou 8) octets de différence entre les deux adresses, ce qui correspond à la taille du type (entier) sur la machine.

TP précédent

Créer un tableau d'entiers de 10 cases et le remplir, puis afficher son contenu.

```
#include <stdio.h>
int main()
{
    //int tab[10] = {0,1,2,3,4,5,6,7,8,9};
    int tab[10];
    for(int i = 0; i < 10; i++)
        tab[i] = i;

    for(int i = 0; i < 10; i++)
        printf("%d ", tab[i]);

    printf("\n");
}
```

TP précédent

Créer une fonction qui prend en paramètre un tableau d'entiers et sa taille, et qui affiche son contenu.

```
#include <stdio.h>
int affiche_tab(int tab[], int n)
{
    for(int i = 0; i < n; i++)
        printf("%d ", tab[i]);

    printf("\n");
}
int main()
{
    int tab[10] = {0,1,2,3,4,5,6,7,8,9};
    affiche_tab(tab, 10);
}
```

Allocation dynamique

- Pourquoi allouer dynamiquement un tableau ?

Allocation dynamique

- Pourquoi allouer dynamiquement un tableau ?

Jusqu'à maintenant, on a alloué nos tableaux statiquement.
Cela signifie qu'on doit connaître la taille du tableau à la compilation, et donc qu'elle ne peut pas dépendre d'un critère lors de l'exécution (par ex. une entrée utilisateur).

L'allocation dynamique permet donc de créer des tableaux à l'exécution, avec une taille non connue à l'avance.
Il y a cependant une autre utilité.

Allocation dynamique

```
#include <stdio.h>
int main()
{
    int *tab = creer_tab();
    for(int i = 0; i < 10; i++)
        tab[i] = i;

    for(int i = 0; i < 10; i++)
        printf("%d ", tab[i]);
    printf("\n");
    return 0;
}
```

Coder la fonction **creer_tab** qui renvoie un tableau de taille 10.

Allocation dynamique

```
#include <stdio.h>
int* creer_tab()
{
    int tab[10];
    return tab;
}

int main()
{
    int *tab = creer_tab();
    for(int i = 0; i < 10; i++)
        tab[i] = i;

    for(int i = 0; i < 10; i++)
        printf("%d ", tab[i]);
    printf("\n");
    return 0;
}
```

Ce code, qui semblerait le plus simple, ne fonctionne pas.
Pourquoi ? Parce que le tableau est crée statiquement au sein de la fonction **creer_tab**, et donc est détruit à la sortie de la fonction (chaque variable statique ne vit que dans son bloc).

L'adresse retournée est donc désallouée, et inutilisable.

Allocation dynamique

- Pourquoi allouer dynamiquement un tableau ?

On peut donc allouer dynamiquement un tableau pour avoir une taille non déterminée à l'avance, ou pour pouvoir gérer la durée de vie de l'allocation, et ce pas uniquement sur les tableaux.

Allocation dynamique

- Comment allouer dynamiquement de la mémoire ?

Allocation dynamique

- Comment allouer dynamiquement de la mémoire ?

Pour cela, on utilise la fonction **malloc** présente dans la bibliothèque **stdlib.h**, qui prend en paramètre le nombre d'octets à réserver, et renvoie l'adresse du début de la zone réservée.

Si on veut allouer suffisamment de mémoire pour un tableau de 10 cases, alors
int *tab = malloc(10 * sizeof(int));

sizeof étant un opérateur unitaire renvoyant la taille du type donné en paramètre.

Remarque Ne pas oublier d'inclure **stdlib.h** avec **#include**.

Allocation dynamique

- Comment allouer dynamiquement de la mémoire ?

Il existe aussi la fonction **calloc** (de la même bibliothèque) qui prend en paramètres le nombre de blocs à réserver ainsi que la taille de ces blocs, et renvoie l'adresse du début de la zone réservée.
À la différence de **malloc**, ces blocs seront initialisés à 0 (alors que **malloc** n'initialise pas).

Si on veut allouer suffisamment de mémoire pour un tableau de 10 cases, alors
int *tab = calloc(10, sizeof(int));

Allocation dynamique

- Comment désallouer de la mémoire dynamiquement allouée ?

Allocation dynamique

- Comment désallouer de la mémoire dynamiquement allouée ?

Comme dit, une mémoire allouée dynamiquement n'est plus conditionnée par la durée de vie du bloc auquel elle appartient. Il convient donc de désallouer nous-même cette mémoire, pour éviter les fuites.

Pour cela, il faut faire appel à la fonction **free** (de la même bibliothèque que précédemment) qui prend en paramètre l'adresse de la zone allouée et qui se charge de libérer cette dernière.

Si on veut désallouer notre tableau de tout à l'heure, alors
free(tab);

TP

Objectifs du TP :

- Créer une fonction qui prend en paramètre une taille et renvoie un tableau d'entiers de cette taille.
- Créer une fonction qui prend en paramètre un tableau d'entiers et une taille, et renvoie un autre tableau avec les mêmes valeurs mais triées par ordre croissant (un algorithme naïf suffira).