Allocation Dynamique

06 | MAR | 2020

Dorian H. Mekni

Allocation ->

- Nous créons un tableau de joueurs
- Nous ne connaissons pas le nombre de joueurs ni la place que chacun va prendre en terme de mémoire ainsi que son adresse dans la RAM.
- Nous utilisons l'allocation dynamique pour s'assurer que la place nécessaire pour chacun des ces joueurs soit disponible à une adresse déterminée par le compilateur sur la mémoire vive

- On peut dynamiquement reserver un emplacement mémoire selon ce que l’utilisateur a rentré
- Cela est utile pour la saisie d’un nom par exemple car nous ne savons pas si le pseudo sera composé de combien de caractères.
- Cette solution est totalement adaptable
-

malloc()

- malloc() alloue un bloc de memoire>
- La mémoire qui nous est allouée dynamiquement n’est pas effacée automatiquement.
- Il convient donc d’utiliser la fonction free() une l’utilisation fini afin de libérer l’espace memoire qui a été alloué.

malloc()

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

int main(void)

{

    int number_of_players = 0;
    int *list_of_players = NULL;
    int i;

    printf("How many players ? ");
    scanf("%d" , &number_of_players);

    list_of_players = malloc(sizeof(int) * number_of_players);
    // un tableau de taille variable est donc crée, vu que l'on ne connait pas sa tailled'emblée, l'allocation dynamique devient une option utile dans ce cas.

    if(list_of_players == NULL)
        exit(1);

    for (i = 0; i < number_of_players ; i++)
    {
        printf("Player %d = numero %d\n", i + 1, i * 3);
        list_of_players[i] = i * 3;
    }

    for(i = 0; i < number_of_players; i++)
    {
        printf("[%d] ", list_of_players[i]);
    }

    free(list_of_players);

    return 0;
}
```

free()

- free() désalloue l’espace mémoire obtenu par malloc(), calloc() et realloc()
- Il n’y a pas de valeur de retour produite par cette dernière.

calloc()

- c pour client
- alloc pour allocation
- 2 paramètres
- On y place la donnée concernée que l'on veut traiter puis on y ajouter la taille: sizeof()
- Elle sert à allouer dynamiquement comme malloc(0 mais en initialisant tout à zéro plutôt que de manière désordonnée.
- On obtient donc des 0 par défaut.

calloc()

```c
int main(void)

{

    int number_of_players = 0;
    int *list_of_players = NULL;
    int i;

    printf("How many players ? ");
    scanf("%d" , &number_of_players);

    list_of_players = calloc(number_of_players, sizeof(int));


    if(list_of_players == NULL)
        exit(1);

    /*for (i = 0; i < number_of_players ; i++)
    {
        printf("Player %d = numero %d\n", i + 1, i * 3);
        list_of_players[i] = i * 3;
    }
    */
    for(i = 0; i < number_of_players; i++)
    {
        printf("[%d] ", list_of_players[i]);
    }

    free(list_of_players);

    return 0;
}
```

```
How many players ? 5
[0] [0] [0] [0] [0] Program ended with exit code: 0
```

realloc() 1

- realloc() reprend l'accès mémoire qui est alloué par malloc
- realloc() permet de reclouer un bloc de memoire dans le tas(heap)
- Dans d'autres termes, si l'espace memoire libre qui suit le bloc de memoire à reclouer contient assez de place, ce bloc est simplement élargi.
- En revanche si l'espace libre ne suffit pas, un nouveau bloc de memoire sera donc alloué.
- Le contenu de la zone d'origine sera transféré dans le nouvel espace memoire de manière automatique.

realloc() 2

- Si le bloc de mémoire ne pourrait être alloue(plus de memoire dispo) le pointeur NULL nous sera délivré.
- Il est nécessaire de tester que la fonction ne retourne pas ce genre de valeur
- TIPS: Il convient d'utiliser <assert>

realloc() 3

```c
    printf("\n.....................................\n");

    number_of_players = 8;

    list_of_players = realloc(list_of_players, number_of_players * sizeof(int));

    if(list_of_players == NULL)
    exit(1);

    for (i = 0; i < number_of_players ; i++)
    {
        printf("Player %d = numero %d\n", i + 1, i * 3);
        list_of_players[i] = i * 3;
    }

    for(i = 0; i < number_of_players; i++)
    {
        printf("[%d] ", list_of_players[i]);
    }


    free(list_of_players);

    return 0;
}
```

```
Player 1 = numero 0
Player 2 = numero 3
Player 3 = numero 6
[0] [3] [6]
.....................................
Player 1 = numero 0
Player 2 = numero 3
Player 3 = numero 6
Player 4 = numero 9
Player 5 = numero 12
Player 6 = numero 15
Player 7 = numero 18
Player 8 = numero 21
[0] [3] [6] [9] [12] [15] [18] [21] Program ended with exit code: 0
```