Allocation dynamique de la mémoire

Introduction

Supposons que vous voulez écrire un programme permettant d'entrer des entiers mais, ce nombre d'entiers à entrer peut changer à chaque fois. C'est-à-dire, parfois vous voulez entrer 10 entiers, une autre fois vous voulez entrer 50 entiers et une autre fois que 5 entiers. Vous n'allez pas changer le programme à chaque fois. Il faut écrire un programme une seule fois et qui s'adapte à chaque situation.

Voici le programme suivant permettant de résoudre ce problème. On suppose dans ce programme qu'on dépasse pas 100 entiers à entrer au clavier.

```
#include <stdio.h>
#define max 100
int main()
{
    int TAB[max] ;
    int nbElement;
    int i;

    printf("Entrez le nombre d'éléments du tableau :");
    scanf("%d",&nbElement);
    for(i=0;i<nbElement; i++)
    {
      printf("TAB[%d]:",i);
      scanf("%d",&TAB[i]);
    }

    for(i=0;i<nbElement; i++)
      printf("%d\t",TAB[i]);
}
```

Dans ce programme nous créons un tableau de 100 éléments de type entier. Voir déclaration int TAB[max], sachant que max est une constante ayant pour valeur 100 : #define max 100

Ensuite, nous entrons le nombre d'éléments à remplir dans ce tableau TAB. C'est-à-dire nous n'allons pas remplir les 100 entiers mais nous précisons d'abord le nombre d'éléments à entrer : nbElement. Puis en fonction de ce nombre nous allons remplir notre tableau TAB.

Pour comprendre mieux, par exemple, vous ne savez pas dés le début, combien d'éléments il faut entrer. Pour cela, vous créez un tableau avec un maximum d'éléments, vous devez savoir combien au max vous pouvez avoir d'éléments, ensuite le programme vous invite à entrer le nombre d'élément à saisir au clavier.

Ce programme présente un inconvénient majeur ! je ne parle pas de la compilation. Le programme fonctionne parfaitement. A ce niveau, nous pensons plus à écrire juste un programme qui marche, mais son optimisation.

Je vous explique

Trois composant dans l'ordinateur, peuvent ralentir l'exécution d'une application :

1. La RAM lorsqu' elle est pleine
2. Le CPU lorsqu'il est occupé
3. Le disque dur lorsqu'il lent et plein

Un programmeur doit prendre en considération ses paramètres lors de l'écriture de son programme. Tout le monde peut écrire un programme mais, il y a toujours un programme meilleur qu'un autre programme grâce à son optimisation.

Par exemple, un programme qui occupe moins de l'espace mémoire dans la RAM est meilleur qu'un autre programme faisant la même chose mais occupant plus d'espace mémoire dans la RAM.

Et ça c'est le cas de notre programme exemple. Notre programme déclare 100 éléments entiers dans la RAM mais il se peut qu'on ait besoin que de 10 entiers. Les 90 éléments restants occupent de l'espace dans la RAM inutilement. C'est de l'espace perdu.

Est-ce qu'il y a une possibilité en C permettant de réserver un espace dans la RAM au moment où nous aurons besoin et lorsqu'on termine l'utilisation de cet espace dans la RAM, nous le libérons. C'est-à-dire, au moment où nous aurons besoin d'une variable nous réservons le nombre d'octets correspondant dans la RAM et lorsqu'on a plus besoin, nous libérons ces octets occupés par cette variable.

Oui c'est possible de gérer ce mécanisme grâce à la gestion dynamique de la mémoire appelé aussi, allocation dynamique de la mémoire.

Comment C permet la gestion dynamique de la mémoire ? Grâce à deux fonctions malloc et free le langage C permet de gérer dynamiquement la RAM.

Fonctions malloc et free

Pour mieux comprendre l'allocation dynamique de la mémoire, nous allons prendre un exemple. Soit le programme suivant :

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.  int *TAB ;
5.  int nbElement;
6.  int i;
7.  printf("Entrez le nombre d'éléments du tableau :");
8.  scanf("%d",&nbElement);
9.  TAB=(int*)  malloc(nbElement *sizeof(int));
10. for(i=0;i<nbElement; i++) {
        printf("tab[%d]:",i);
        scanf("%d",&TAB[i]);
    }
11. for(i=0;i<nbElement; i++)
        printf("%d\t",TAB[i]);
12. }
```

Dans l'instruction 4, nous déclarons un pointeur nommé TAB.

Dans l'instruction 5, nous déclarons la variable nbElement qui sert à créer le nombre d'éléments dans la RAM.

Dans l'instruction 6, nous déclarons un indice pour parcourir nos éléments.

Dans l'instruction 7 et 8, nous entrons le nombre d'éléments qu'on aura besoin.

Dans l'instruction 9, nous faisons appel à la fonction malloc. La fonction malloc permet de réserver dans la RAM un certain nombre d'octets précisé entre parenthèse.

Par exemple que veut dire cette instruction : `malloc (sizeof(int))`

la fonction `sizeof(int)` nous retourne le nombre d'octet du type int. C'est-à-dire 4, ainsi `malloc(4)` va réserver 4 octets dans la RAM.

Pour réserver un espace de 5 entiers, nous pouvons écrire : `malloc (5 * sizeof(int))`

Pour notre exemple, nous avons écrit :

```
TAB=(int*)  malloc(nbElement *sizeof(int));
```

Cette instruction permet de créer un tableau ayant pour nombre d'élément le nombre d'élément entré au clavier depuis la variable nbElement. Le tableau n'existe pas auparavant, il est créé au moment où nous précisons le nombre d'éléments à entrer. A chaque exécution, le nombre d'élément du tableau peut être variable.

La fonction malloc va réserver un nombre d'octets en fonction de la variable nbElement entrée au clavier :

```
malloc(nbElement * sizeof(int))
```

Si vous tapez 5, malloc va réserver 5 * 4 octets c'est 20 octets dans la RAM

Maintenant nous souhaitons avoir l'adresse de premier octet de l'espace réservé :

```
(int*) malloc(nbElement *sizeof(int));
```

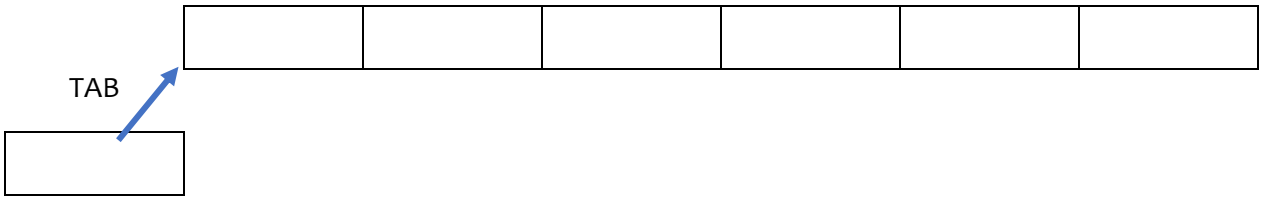
(int *) avant malloc permet d'extraire l'adresse du premier octet de l'espace réservé.

Ensuite nous voulons affecter cette adresse au pointeur TAB :

```
TAB = (int*) malloc(nbElement *sizeof(int));
```

Comme vous le constatez, la fonction malloc, permet de réserver un espace dans la RAM en fonction d'une variable entrée au clavier.

Par exemple, si nous avons tapé 5 pour la variable nbElement, au niveau de la RAM, nous aurons :



Pour parcourir l'espace réservé, vous pouvez utiliser la variable TAB comme un tableau TAB[] ou comme un pointeur. Vous avez le choix.

L'instruction 10, permet de remplir le tableau TAB

L'instruction 11, permet d'afficher le tableau TAB. Dans l'instruction 10 et 11 nous considérons l'espace réservé comme un tableau.

Pour conclure, ce programme permet de créer un tableau de façon dynamique. En fonction de la variable entrée au clavier, nous réservons de l'espace utile. Ainsi, de cette façon, nous gérons dynamiquement la RAM.

La fonction free

La fonction free permet de libérer l'espace créé par la fonction malloc. Vous pouvez utiliser cette fonction lorsque votre programme n'a plus besoin de l'espace réservé et que le programme est toujours en cour d'exécution. Pour notre exemple, nous n'avons pas besoin de la fonction free, pourquoi ? parce que, ce n'est pas intéressant de la mettre en fin du programme. Dans tous les cas toutes les variables vont être libérées de la RAM à la fin du programme.

Pour utiliser la fonction free, voici s syntaxe :

```
free(pointeur) ;
```