Algorithmique et programmation en C - L'allocation dynamique.

François Delbot

Maître de conférences de l'Université Paris Ouest Nanterre la Défense
Membre de l'équipe de recherche opérationnelle du LIP6

April 5, 2018

Sommaire

1 Préambule

2 Représentation de la mémoire

3 Les bases de l'allocation dynamique

4 La ré-allocation de mémoire

5 Allocation dynamique de tableaux à deux dimensions

Prérequis

1. Comprendre et maitriser la portée des variables.
2. Comprendre et maitriser les tableaux.
3. Avoir suivi et compris le cours sur les pointeurs.

1 Préambule

2 Représentation de la mémoire

3 Les bases de l'allocation dynamique

4 La ré-allocation de mémoire

5 Allocation dynamique de tableaux à deux dimensions

Sommaire

1 Préambule

2 Représentation de la mémoire

3 Les bases de l'allocation dynamique

4 La ré-allocation de mémoire

5 Allocation dynamique de tableaux à deux dimensions

Représentation de la mémoire

Les différentes stratégies d'allocation de la mémoire

Un programme consomme de la mémoire :

1. Des variables globales
2. Des variables statiques
3. Des fonctions

Nous allons survoler les méthodes pour enregistrer en mémoire ces différentes données.

Représentation de la mémoire
Allocation statique

Stratégie d'allocation statique

- Une variable globale existe durant toute l'exécution du programme. On connait donc la quantité exacte de mémoire nécessaire dès la compilation.
- Une variable statique est une variable locale dont le contenu persiste entre deux appels de fonctions. Tout comme les variables globales, la quantité de mémoire nécessaire est constante durant toute l'exécution du programme.

Dans les deux cas, la réservation se fait lors de l'initialisation du programme, juste avant son exécution. Cette méthode est très rapide.

Représentation de la mémoire

Allocation (dynamique) automatique

Stratégie d'allocation (dynamique) automatique

- Dans certains cas, l'ordinateur se charge de réserver pour vous de la mémoire.
- A la fin de la portée de ces données, c'est l'ordinateur qui se charge de libérer cette mémoire automatiquement.

Par exemple, les variables locales lors des appels de fonction ....
Cette méthode est très pratique puisque cela évite les fuites de mémoire.

Représentation de la mémoire

Allocation (dynamique) automatique. Appel de fonction.

Lorsqu'on appelle une fonction, la fonction courante (dite appelante) est mise en pause :

1. On enregistre l'adresse où reprendre après l'exécution de la fonction appelée.
2. On réserve un espace dans la mémoire pour enregistrer le résultat.

De plus, l'appel à une fonction déclenche automatiquement la réservation de mémoire pour un certain nombre de choses :

1. Une variable contenant le nombre de paramètres
2. Les paramètres formels
3. Les variables locales

Une fois la fonction terminée, cette mémoire est automatiquement libérée.

Représentation de la mémoire
Allocation dynamique manuelle

Stratégie d'allocation dynamique manuelle

- Dans certains cas, on ne connait la quantité de mémoire nécessaire que durant l'exécution du programme.
- Il est possible de demander explicitement (via un appel de fonction) la réservation d'un espace en mémoire.
- Lorsque ces données ne sont plus utilisée, il est impératif de libérer cette mémoire (via un appel de fonction).

1. Permet le contrôle total de l'allocation et de la libération.
2. Très souple.
3. Durée de vie indéfinie.
4. Contraignant : le programmeur a la responsabilité de la libération.

Représentation de la mémoire

Différents usages, différentes portées.

On peut distinguer différents usages de la mémoire :

- La liste des instructions du programme est fixe.
- Les variables globales sont connues dans tout le programme, du début à la fin de son exécution. La quantité de mémoire utilisée est constante.
- Certaines variables ne sont nécessaires que temporairement. C'est le cas des variables locales, par exemple.
- Un appel de fonction consomme de la mémoire jusqu'à la fin de son exécution.
- On peut demander de la mémoire ponctuellement, en fonction de paramètres non connus à l'avance.

Pour faciliter le développement et minimiser la quantité de mémoire utilisée, elle est divisée en différents groupes fonctionnels.

Représentation de la mémoire

Groupes fonctionnels

1. **Le programme :** contient la suite des instructions à exécuter
2. **Les données et les constantes :** contient les variables globales et les variables statiques
3. **La pile :** contient la mémoire allouée automatiquement
4. **Le tas :** contient la mémoire allouée manuellement

Représentation de la mémoire
Bilan

1. La mémoire d'un programme forme un bloc d'octets contigus.
2. Elle est séparée en différents groupes fonctionnels.
3. Les données d'un programme ne sont pas toutes enregistrées dans le même groupe fonctionnel. Cela dépends de leur portée et de leur mode de réservation.
4. Le tas est une zone mémoire de taille importante permettant de réaliser manuellement des allocations de mémoire.

Représentation de la mémoire

Comparaison de ces différentes méthodes d'allocation

Exercice : remplissez ce tableau

	Allocation statique	Allocation automatique	Allocation dynamique
Avantages	?	?	?
Inconvénients	?	?	?

Sommaire

1 Préambule

2 Représentation de la mémoire

3 Les bases de l'allocation dynamique

4 La ré-allocation de mémoire

5 Allocation dynamique de tableaux à deux dimensions

L'allocation dynamique de mémoire

Pour le moment, lorsque nous écrivons un programme nous devons connaitre à l'avance le nombre de variables que nous allons utiliser (et les tailles exactes de nos tableaux) ainsi que leur durée de vie (qui dépends de l'endroit et de la manière dont on les déclares).

L'allocation dynamique de mémoire

Problèmes rencontrés

Dans une fonction, on souhaite afficher la liste des noms des étudiants d'une promotion :

```
1  void affichage()
2  {
3      char tab[1000][255];
4      int i,nb;
5      printf("Entrez le nombre d'etudiants :");
6      scanf("%d",&nb);
7      for(i=0;i<nb;i++)
8      {
9          scanf("%s",tab[i]);
10     }
11     printf("Affichage des noms des etudiants :\n");
12     for(i=0;i<nb;i++)
13     {
14         printf("\nEleve %d : %s",i,tab[i]);
15     }
16 }
```

1. Que se passe-t-il si le nombre d'étudiants est faible ?
2. Que se passe-t-il si le nombre d'étudiants est très important ?

L'allocation dynamique de mémoire

Problèmes rencontrés

1. On risque de consommer trop de mémoire par rapport au faible nombre de données.
2. On risque de déborder de la mémoire si la quantité de données est trop importante.
3. On ne peut pas retourner le tableau car la mémoire est automatiquement libérée à la fin de la fonction.
4. Chaque modification de la taille du tableau implique une recompilation.

L'allocation dynamique de mémoire

Problèmes rencontrés

L'allocation dynamique de mémoire va nous permettre de régler ces problèmes. Nous allons demander au système, au cours de l'exécution du programme, de nous réserver en mémoire un espace correspondant à la quantité de donnée que nous devons manipuler. Ainsi, pas de mémoire inutilisée, de débordement de tableau causé par une valeur saisie par l'utilisateur ni de nécessité de recompiler notre programme. Formidable non ?

Allocation de mémoire
La fonction malloc

La fonction malloc

La fonction malloc permet de réserver un espace en mémoire. La taille de cet espace doit être exprimée en octets. La fonction retourne l'adresse du début du bloc qui aura été réservé. Si l'opération d'allocation échoue, la valeur NULL est retournée.

```
1   void *malloc(size_t size);
```

Attention :

1. La fonction malloc n'initialise pas le contenu de l'espace mémoire réservé. Cet espace risque donc de contenir des valeurs résiduelles.
2. Le type size_t correspond à un entier non signé.
3. La fonction malloc retourne un pointeur de type void *.
4. Si la réservation de mémoire échoue, la fonction retourne NULL.

Allocation de mémoire

La fonction malloc

Bonne pratique

Après un appel à la fonction malloc, vous devez impérativement vérifier si l'allocation s'est bien déroulée. Si l'adresse retournée vaut NULL, c'est qu'un problème a eu lieu et que la mémoire n'a pas été allouée. Lire ou écrire à l'adresse NULL ferait planter le programme, vous ne devez surtout pas tenter d'y accéder.

Allocation de mémoire

Type de retour de la fonction malloc: void*

- La fonction malloc va réserver un espace d'un certain nombre d'octets en mémoire.
- Cet espace mémoire n'est pas typé en lui même.
- C'est la manière dont on va lire et écrire dans ce bloc mémoire qui va donner son type aux données.

Par exemple, si on réserve 4 octets par la fonction malloc, s'agit-il d'un espace pour un int ou d'un espace pour enregistrer 4 char ?

Allocation de mémoire

Opération de transtypage

Malloc retourne une adresse dont on ne connait pas le type, ce qui est représenté par le type void *. On va « transformer » ce type par défaut en le type de données que l'on va manipuler dans cet espace.

Voici un exemple complet :

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void)
5  {
6    int *ptr;
7    ptr=(int *)malloc(sizeof(int));
8    if(ptr==NULL)
9    {
10       printf("Erreur lors de l'allocation memoire !");
11       exit(EXIT_FAILURE);
12   }
13   printf("Allocation memoire reussie ! Adresse du premier octet de cet espace :
          %p",ptr);
14   return EXIT_SUCCESS;
15 }
```

Allocation de mémoire
La fonction calloc

La fonction calloc

La fonction calloc permet de réserver un espace en mémoire et l'initialiser avec des 0.

Cette fonction accepte deux arguments :

1. le nombre de variables que l'on souhaite réserver
2. la taille en octets occupée par chacune de ces variables.

Elle retourne un pointeur vers le premier octet de l'espace réservé, NULL est retourné si l'opération échoue.

Allocation de mémoire

La fonction calloc. Exemple

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void)
5  {
6    int *ptr;
7    ptr=(int *)calloc(1,sizeof(int));
8    if(ptr==NULL)
9    {
10       printf("Erreur lors de l'allocation memoire !");
11       exit(EXIT_FAILURE);
12   }
13   printf("Allocation memoire reussie ! Adresse du premier octet de cet espace :
           %p",ptr);
14   return EXIT_SUCCESS;
15 }
```

Allocation de mémoire

Différences entre malloc et calloc

1. La fonction malloc est plus rapide que la fonction calloc.
2. La fonction calloc initialise la mémoire à 0, ce qui n'est pas le cas de la fonction malloc.
3. La fonction calloc est plus rapide que la fonction malloc suivie d'une boucle pour initialiser la mémoire.

Libération de la mémoire

1. La mémoire allouée dynamiquement va rester réservée durant toute l'exécution du programme.
2. Il est raisonnable de ne pas conserver de mémoire non utilisée en la libérant.
3. La libération de mémoire est TRÈS importante pour éviter des fuites de mémoire ou le swap.

Libération de la mémoire
La fonction free

La fonction free

La fonction free permet de libérer un espace mémoire réservé au préalable. Pour libérer cet espace, il suffit d'indiquer l'adresse du début du bloc à libérer.

```
1  void free(void *ptr);
```

Libération de la mémoire

La fonction free. Exemple.

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void)
5  {
6    int *ptr;
7    ptr=(int *)malloc(sizeof(int));
8    if(ptr==NULL)
9    {
10       printf("\nErreur lors de l'allocation memoire !");
11       exit(EXIT_FAILURE);
12   }
13   printf("\nAllocation memoire reussie ! Adresse du premier octet de cet espace
         : %p",ptr);
14   free(ptr);
15   printf("\nLa memoire allouee vient d'ete liberee !\n");
16   return EXIT_SUCCESS;
17 }
```

Libération de la mémoire

La fonction free

Attention : la fonction free ne doit être appelée que si la mémoire à bien été allouée. Si on essaye de libérer de la mémoire qui n'est pas allouée, cela risque (fortement) de produire une erreur et de faire planter votre programme.

Erreurs très fréquentes :

1. Ne pas libérer la mémoire et provoquer une fuite de mémoire
2. Libérer deux fois la mémoire

Allocation dynamique et tableaux

Retour sur les tableaux

Un tableau est une suite d'éléments d'un même type placés de manière contiguë en mémoire.

La phrase magique

Le nom d'un tableau est un pointeur sur son premier élément.

Allocation dynamique et tableaux

Les fonctions malloc et calloc retournent un pointeur vers le premier octet de la zone mémoire allouée. Cette zone est donc manipulable grâce à ce pointeur. Exemple :

```
1     int *ptr;
2     ptr=(int *)malloc(sizeof(int)*1000);
3     if(ptr==NULL)
4     {
5         printf("Erreur lors de l'allocation memoire !");
6         exit(EXIT_FAILURE);
7     }
```

Pour affecter une valeur au premier espace pouvant contenir un entier (dans les 4 premiers octets), on peut écrire :

```
1     *ptr = 5;
```

Pour affecter une valeur au deuxième espace pouvant contenir un entier (dans les 4 octets suivants), on peut écrire :

```
1     *(ptr+1) = 7;
```

Or, en langage C, il y a équivalence entre les notations *(ptr+i) et ptr[i]. On peut donc manipuler le pointeur ptr comme un tableau !

Allocation dynamique et tableaux

Retour sur les fonctions d'allocation

```c
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    int *ptr;
    int nb,i;
    printf("Entrez une valeur :");
    scanf("%d",&nb);
    ptr=(int *)malloc(sizeof(int)*nb);
    if(ptr==NULL)
    {
        printf("Erreur lors de l'allocation memoire !");
        exit(EXIT_FAILURE);
    }
    printf("Allocation memoire reussie ! Adresse du 1er octet : %p",ptr);
    for(i=0; i<nb; i++)
    {
        ptr[i]=i%10;
    }
    for(i=0; i<nb; i++)
    {
        printf("%2d",ptr[i]);
    }
    free(ptr);
    return EXIT_SUCCESS;
}
```

Sommaire

1 Préambule

2 Représentation de la mémoire

3 Les bases de l'allocation dynamique

4 La ré-allocation de mémoire

5 Allocation dynamique de tableaux à deux dimensions

Besoin de redimensionner la mémoire déjà allouée

- L'allocation dynamique permet de réserver de l'espace mémoire même si nous ne connaissons pas la quantité nécessaire avant l'exécution de notre programme.
- Cependant, nous avons considéré qu'il existe un certain moment de notre programme à partir duquel nous avons connaissance de la quantité de mémoire à réserver.
- Mais supposons un instant que la quantité de mémoire nécessaire à la bonne exécution de notre programme change régulièrement. Comment allons nous faire ?

Besoin de redimensionner la mémoire déjà allouée

Idée. A ne pas faire ...

On pourrait allouer dynamiquement un nouveau tableau à la bonne taille, recopier les données de l'ancien tableau vers le nouveau, puis libérer l'ancien tableau.

Besoin de redimensionner la mémoire déjà allouée
La fonction realloc

La fonction realloc

La fonction realloc permet de changer la taille du bloc mémoire alloué pointé par ptr en le redimensionnant à size octets .

- Si on diminue la taille du bloc mémoire : Le contenu est inchangé pour l'ensemble du bloc restant.
- Si on augmente la taille du bloc : la mémoire ajoutée n'est pas initialisée.
- La fonction realloc retourne un pointeur vers le nouvel espace mémoire alloué.
- Si la fonction realloc echoue à modifier la taille du bloc, le bloc original n'est pas modifié, ni libéré.

Besoin de redimensionner la mémoire déjà allouée
La fonction realloc

La fonction realloc

La fonction realloc permet de changer la taille du bloc mémoire alloué pointé par ptr en le redimensionnant à size octets .

```
1     void *realloc(void *ptr, size_t size);
```

Sommaire

1 Préambule

2 Représentation de la mémoire

3 Les bases de l'allocation dynamique

4 La ré-allocation de mémoire

5 Allocation dynamique de tableaux à deux dimensions

Allocation dynamique de tableaux à deux dimensions

Exemple d'allocations multiples

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void)
5  {
6      int *tab1,*tab2,*tab3,*tab4,*tab5;
7      int nb=10;
8      tab1=(int *)malloc(sizeof(int)*nb);
9      tab2=(int *)malloc(sizeof(int)*nb);
10     tab3=(int *)malloc(sizeof(int)*nb);
11     tab4=(int *)malloc(sizeof(int)*nb);
12     tab5=(int *)malloc(sizeof(int)*nb);
13
14     return EXIT_SUCCESS;
15 }
```

Allocation dynamique de tableaux à deux dimensions

Exemple d'allocations multiples

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void)
5  {
6      int * tab[5];
7      int nb=10;
8      tab[0]=(int*) malloc(sizeof(int)*nb);
9      tab[1]=(int*) malloc(sizeof(int)*nb);
10     tab[2]=(int*) malloc(sizeof(int)*nb);
11     tab[3]=(int*) malloc(sizeof(int)*nb);
12     tab[4]=(int*) malloc(sizeof(int)*nb);
13
14     return EXIT_SUCCESS;
15 }
```

Allocation dynamique de tableaux à deux dimensions

Exemple d'allocations multiples

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   int main(void)
5   {
6       int ** tab;
7       int nb=10;
8
9       tab=(int**) malloc(sizeof(int*)*5);
10
11      tab[0]=(int*) malloc(sizeof(int)*nb);
12      tab[1]=(int*) malloc(sizeof(int)*nb);
13      tab[2]=(int*) malloc(sizeof(int)*nb);
14      tab[3]=(int*) malloc(sizeof(int)*nb);
15      tab[4]=(int*) malloc(sizeof(int)*nb);
16
17      return EXIT_SUCCESS;
18  }
```

Allocation dynamique de tableaux à deux dimensions

Exemple d'allocations multiples

```
1   #include <stdio.h>
2   #include <stdlib.h>
3
4   int main(void)
5   {
6       int i;
7       int ** tab;
8       int nb=10;
9
10      tab=(int**) malloc(sizeof(int*)*5);
11      for(i=0;i<5;i++)
12      {
13              tab[i]=(int*) malloc(sizeof(int)*nb);
14      }
15      return EXIT_SUCCESS;
16  }
```

Allocation dynamique de tableaux à deux dimensions

Exemple d'allocations multiples

Exercice : Implémentez le calcul du triangle de Pascal. L'utilisateur doit vous indiquer le nombre de lignes du triangle. Vous devez allouer la mémoire du triangle de manière dynamique.