Allocation dynamique de la mémoire et Structures en C

Ferdinand KAHENGA

UDBL

Mai 2025

Plan du chapitre

- Allocation de la mémoire
  1. Problème de taille des tableaux
  2. malloc
  3. free
  4. calloc
- Structures en C
  1. Déclaration d'une structure
  2. Utilisation d'une structure

Problème de taille des tableaux

- Lorsque l'on déclare un tableau, il est obligatoire de préciser sa taille. Cela signifie que la taille d'un tableau doit être connue à la compilation.
- Que faire si on ne connait pas cette taille ? La seule solution qui se présente pour le moment est le surdimensionnement, on donne au tableau une taille (trop) élevée de sorte qu'aucun débordement ne se produise.
- Nous aimerions procéder autrement, c-à-d préciser la dimension du tableau au moment de l'exécution

Fonction malloc

- Lorsque vous déclarez un pointeur *int *p*, vous allouez un espace mémoire pour y stocker l'adresse mémoire d'un entier
- Et p, jusqu'à ce qu'on l'initialise, contient n'importe quoi. Vous pouvez ensuite faire pointer p sur l'adresse mémoire que vous voulez (de préférence une zone contenant un int...).
- Soit cette adresse est celle d'une variable qui existe ou soit cette adresse est celle d'un espace mémoire créé spécialement pour l'occasion

Fonction malloc

- En cas de besoin, pouvez demander au Système d'exploitation de l'espace mémoire.
- La fonction qui permet de réserver n octets est *malloc(n)*. une allocation dynamique, c-a-d une allocation de la mémoire au cours de l'exécution.
- Pour stocker un int par exemple, il suffit d'appeler malloc(4)
- *malloc* retourne l'adresse mémoire du premier octet de la zone réservée. Par conséquent, si vous voulez créer un int, il convient d'exécuter l'instruction :p = malloc(4)
  ou p est de type int *.

Fonction malloc

- L'instruction p = malloc(4) ne peut pas passer à la compilation. Le compilateur vous dira que les types void* et int* sont incompatibles (incompatible types in assignement).
- Pour ce faire, il convient d'exécuter un cast. Dans l'exemple ci-avant, cela donne :
  *int * p = (int *) malloc(4)*

Exemple du code

```c
int main()
{
    int *p=NULL;
    p=(int *)malloc(sizeof(int));
    *p=8;
    printf("%d",*p);
    return 0;
}
```

Vous remarquez que l'on connait l'adresse d'une variable mais pas son nom.
Le seul moyen de manier la variable allouée dynamiquement est d'utiliser un pointeur.

Fonction malloc

- La fonction malloc retourne NULL si aucune zone mémoire adéquate n'est trouvée.
- Il convient, à chaque malloc, de vérifer si la valeur retournée par malloc est différente de NULL

Exemple du code

```c
int *p=NULL;
p=(int *)malloc(sizeof(int));
if(p==NULL)
{
  printf("Un probleme de memoire se pose!.\n");
  exit(0);
}

*p=8;
printf("%d",*p);
free(p);
```

Fonction free

- Lorsqu'on exécute une allocation dynamique, l'espace réservé ne peut pas être alloué pour une autre variable.
- Une fois que vous n'en avez plus besoin, vous devez le libérer explicitement si vous souhaitez qu'une autre variable puisse y être stockée.
- La fonction de libération de la mémoire est *free*.
- free(p) où p est une variable contenant l'adresse mémoire de la zone a libérer.
- A chaque fois que vous allouez une zone mémoire, vous devez la libérer !
- Voir l'exemple d'utilisation au slide précédent

Fonction calloc

- Elle alloue n blocs de taille t. elle est donc presque équivalente à malloc
- La seule différence réside dans le contenu des cases qui sont allouées
- Avec malloc(), le contenu est totalement aléatoire tandis qu'avec calloc, les cases contiennent des valeurs nulles (tous les bits du bloc alloué sont mis à 0).
- Ceci est très utile pour initialiser rapidement un tableau de nombre.

Tableau avec malloc

```c
int *p=NULL;
int taille=10;
p=(int *)malloc(taille*sizeof(int));
//Remplissage des casses
int i;
for(i=0;i<taille;i++)
{
    *(p+i)=i;
}
//Affichage les éléments du tableau
int *pp=NULL;
for(pp=p;pp<p+taille;pp++)
{
    printf("%d\n",*pp);
}
```

Structures en C

- Nous avons vu qu'un tableau permettait de désigner sous un seul nom plusieurs valeurs de même type.
- Une structure permet de désigner sous un seul nom un ensemble de valeurs pouvant être de types différents
- Un élément de la structure est nommé *champ*
- L'accès à chaque champ se fera, cette fois, non plus par une indication de position, mais par son nom au sein de la structure

Déclaration d'une structure

```
struct personne{
    char nom[50];
    char postnom[50];
    int age;
};
```

- La présence du mot clé **struct** indique la déclaration d'une structure
- Nous avons déclaré une structure nommée **personne**.
- Les champs de notre structure sont :*nom*, *postnom*, *age*
- *nom et postnom* sont en chaînes des caractères et *age* en entier.

Utilisation d'une structure

```c
#include <stdio.h>
#include <stdlib.h>
struct personne{
    char nom[50];
    char postnom[50];
    int age;
};
int main()
{

    struct personne p1={"KAZADI","MUKENDI",30};
    struct personne p2;
    printf("Saisir les valeurs de p2:\n");
    scanf("%s",p2.nom);
    scanf("%s",p2.postnom);
    scanf("%d",&p2.age);
    printf("==================================\n");
    printf("\t Valeurs de p1\n");
    printf("Nom:%s\n",p1.nom);
    printf("Post-Nom:%s\n",p1.postnom);
    printf("Age:%d\n",p1.age);
    printf("==================================\n");
    printf("\t Valeurs de p2\n");
    printf("Nom:%s\n",p2.nom);
    printf("Post-Nom:%s\n",p2.postnom);
    printf("Age:%d\n",p2.age);
    return 0;
}
```

```
"F:\Documents G1\Cours C\2019_2020\UKA\Exemples\Exe
Saisir les valeurs de p2:
KAHENGA
NGONGO
30
==================================
        Valeurs de p1
Nom:KAZADI
Post-Nom:MUKENDI
Age:30
==================================
        Valeurs de p2
Nom:KAHENGA
Post-Nom:NGONGO
Age:30
```

Explication

- Pour utiliser la structure personne déjà déclarée, il suffit d'indiquer dans le main
  struct personne p1 ;
- *struct* est un mot clé du langage
- *personne* est le nom de la structure
- *p1* est le nom de la variable à utiliser.
- *p1* est comme une instance du modèle *personne*
- Par principe chaque instance d'un modèle possède les différents attributs de ce modèle
- On peut avoir plusieurs instances du même modèle, dans notre cas *p2*

Initialisation d'une structure

- On peut initialiser une structure de 2 façons, soit en ajoutant directement des valeurs lors de la déclaration ou soit en affectant individuellement une valeur à chaque champ.
- Dans notre exemple, p1 a été initialisé directement et p2 par l'initialisation individuelle des champs.

Tableau des structures

Écrire un programme qui doit demander les identités de 5 personnes et les afficher en utilisant la structure personne déjà déclarée.

Tableau des structures

```c
#include <stdio.h>
#include <stdlib.h>
struct personne{
    char nom[50];
    char postnom[50];
    int age;
};
int main()
{

    struct personne personnes[5];
    int i;
    for(i=0;i<5;i++)
    {
        struct personne p;
        printf(">> ");
        scanf("%s %s %d",p.nom,p.postnom,&p.age);
        personnes[i]=p;
    }
    for(i=0;i<5;i++)
    {
        printf("=====================================
        printf("Noms: %s -",personnes[i].nom);
        printf("%s \t",personnes[i].postnom);
        printf("Age :%d\n",personnes[i].age);
    }

    return 0;
}
```

```
"F:\Documents G1\Cours C\2019_2020\UKA\Exemples\ExempleCPlus\Structu
>> DIBWE KATALA 15
>> NADIN YANNIC 25
>> ASTRI YOMBO  45
>> ARIEL ARIEL 16
>> JEAN ERICK 15
=======================================
Noms: DIBWE - KATALA    Age :15
=======================================
Noms: NADIN - YANNIC    Age :25
=======================================
Noms: ASTRI - YOMBO     Age :45
=======================================
Noms: ARIEL - ARIEL     Age :16
=======================================
Noms: JEAN - ERICK      Age :15
```

Explication

- A la ligne 11 nous déclarons un tableau des structures.
- Ce tableau doit contenir 5 objets de type *struct personne*.
- A la ligne 15, on déclare une variable temporaire *p* de type *struct personne*.
- Lors de chaque passage dans la boucle, on ajoute une structure p à notre tableau à la position *i* (ligne 18).