INF155 – SÉANCE 11
L'ALLOCATION DYNAMIQUE

Anis Boubaker, Ph.D.
Maître d'enseignement
École de Technologie Supérieure

ÉTS

1

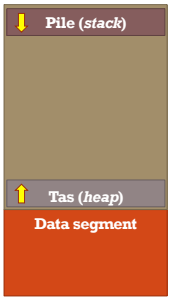
PLAN DE LA SÉANCE

- Retour sur l'allocation mémoire
- Enfin le tas: allouer la mémoire dynamiquement (malloc, calloc, realloc, free)
- Cas d'utilisation: les tableaux dynamiques
- Cas d'utilisation: Les chaines de caractères dynamiques
- Cas d'utilisation: Les enregistrements dynamiques

2

ALLOCATION MÉMOIRE

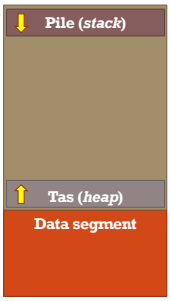
- Il existe deux façons d'allouer la mémoire en C:
  - De façon **statique** – prédéfinie à la compilation
  - De façon **dynamique** - lors de l'exécution selon les besoins du programme (définis manuellement par le programmeur)



3

ALLOCATION STATIQUE

- Jusqu'à maintenant, nous avons laissé le compilateur réserver pour nous l'espace mémoire pour chacune de nos variables;
- Le compilateur détermine par lui-même où allouer un espace mémoire et quelle sera sa durée de vie, en se basant sur les classes de stockage (auto, register, extern, static)
- Données stockées dans la **Pile** ou dans le **Data Segment**
- La taille des données doit être fixée d'avance (à la compilation).
  - Ex.: Nous devons réserver un tableau d'une taille suffisamment grande pour accommoder tous les cas d'utilisation.

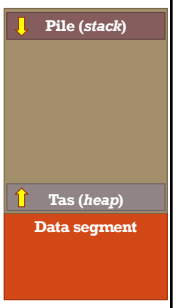


4 ÉTS

4

ALLOCATION DYNAMIQUE

- Contrairement à l'allocation statique, l'allocation dynamique consiste à réserver des espaces mémoire sans que la taille ne soit définie d'avance : c'est décidé à l'exécution.
- L'allocation est faite manuellement (des instructions d'allocation mémoire) afin de réserver des espaces mémoire selon le besoin.
- Les données sont **stockées dans le Tas** et:
  - **Durée de vie:** celle du programme ou jusqu'à ce qu'on libère la mémoire réservée
  - **Visibilité:** Tous le programme, mais il faut connaitre l'adresse (pointeur)



5 ÉTS

5

ALLOCATION DYNAMIQUE – CAS D'EXEMPLE

- On souhaite construire un tableau de distances entre des villes;
- **Méthode statique:** On limite le nombre de villes à MAX_VILLES, et on déclare un tableau de MAX_VILLES x MAX_VILLES
- Problème: On alloue MAX_VILLES^2 cases. Si MAX_VILLES = 20, on alloue 400 doubles (i.e. 3200 octets).

  Si on n'utilise réellement que 3 villes, on a gaspillé 3128 octets!
- Solution: Allocation dynamique!

6 ÉTS

6

ALLOCATION DYNAMIQUE

- Pour allouer dynamiquement de la mémoire, on a besoin de fonctions d'allocation qui sont définies dans la librairie <stdlib.h>
  - Allouer de la mémoire: **malloc** et **calloc**
  - Modifier la taille d'une mémoire allouée: **realloc**
  - Libérer une mémoire allouée: **free**

7

MALLOC ET CALLOC

- Les fonctions d'allocation mémoire permettent de réserver un espace mémoire dans le Tas, de la taille en octets spécifiée.
- La mémoire réservée est une mémoire brute: on peut y stocker ce qu'on veut!
- Les fonctions **malloc** et **calloc** retournent un pointeur vers le premier octet de la zone mémoire réservée

8

VOID*

- **malloc** et **calloc** retournent un pointeur vers une zone mémoire où on peut stocker ce que l'on veut.
- Puisque on peut y stocker ce qu'on veut, le pointeur n'est pas typé (i.e. int*, char*, etc.). C'est un pointeur non-typé ou de type **void***
- **En pratique** : On va toujours trans-typer (cast) le pointeur que l'on obtient dans le type désiré.

9

MALLOC

- **Prototype:**

  void *malloc(size_t taille);

  //Note: size_t est un type entier non-signé

- Exemple: réserver un espace pour stocker un entier:

```
int *entier_d;

entier_d = (int*)malloc(sizeof(int));
*entier_d = 10;
```

10 ÉTS

10

CALLOC

- Variante de **malloc** qui initialise la mémoire de zéros après l'avoir réservée.
- **Prototype:**

  void *calloc(int nb_elems, size_t taille);

- Exemple: réserver un espace pour stocker 10 entiers:

```
int *entier_d;
entier_d = (int*)calloc(10, sizeof(int));
*entier_d = 155; //On stocke la valeur 155 pour le 1er entier
*(entier_d+1) = 33; //On stocke la valeur 33 pour le 2ème entier
```

11 ÉTS

11

MALLOC VS CALLOC

- **calloc** n'offre pas beaucoup d'avantages autres que l'initialisation de la mémoire à 0.
- Cette initialisation a un coût (une boucle).
- Si l'initialisation est importante, on utilise **calloc**. Sinon, on utilise **malloc** (**malloc** dans la très grande majorité des cas!)
- Il est possible d'initialiser la mémoire après l'avoir réservée en utilisant **memset** (RTFM).

12 ÉTS

12

REALLOC

- Il se peut que, après avoir réservé un espace mémoire, nous souhaitions augmenter ou réduire l'espace en question.
- La fonction **realloc** permet de réaliser ce traitement
- **Prototype:**

  void *realloc(void *ptr, size_t nouvelle_taille);

```
int *entier_d;

entier_d = (int*)calloc(1, sizeof(int));
*entier_d = 10;
//Maintenant on veut maintenant avoir 100 entiers
//à cette adresse
entier_d = (int*)realloc(entier_d,100* sizeof(int));
```


13

REALLOC

- La fonction **realloc** renvoie l'adresse de l'espace mémoire réalloué (réduit ou agrandit). L'allocation ne se fera **pas toujours** au même endroit, surtout lorsqu'il s'agit d'agrandir l'espace.
- Il faut donc toujours utiliser la nouvelle adresse retournée par **realloc**.

```
int *entier_d, *nouvelle_adr;
entier_d = (int*)calloc(1, sizeof(int));
realloc(entier_d,100* sizeof(int)); // FAUX!!
nouvelle_adr = (int*)realloc(entier_d,100* sizeof(int));
if(nouvelle_adr!=NULL)
{
        entier_d = nouvelle_adr;
}
```


14

LIMITE DE LA MÉMOIRE

- Les fonctions **malloc**, **calloc** et **realloc** créent un espace mémoire dans le tas.
- En cas de dépassement mémoire, ces fonctions retournent un pointeur **NULL**.
- Il faut toujours vérifier que l'allocation a bien fonctionné:

```
int *entier_d;

entier_d = (int*)malloc(sizeof(int));
assert(entier_d!=NULL); //Vérification de l'allocation
```


15

LIMITE DE LA MÉMOIRE

- Tout espace mémoire réservé avec **malloc**, **calloc** ou **realloc** restera réservé jusqu'à ce qu'un des deux événements se produise:
  - Le programme se termine, ou
  - La mémoire est libérée
- **Ne pas libérer la mémoire qu'on n'utilise pas nous fait courir le risque de dépasser la mémoire disponible et de faire planter le programme (voir le système)**

16

LIMITE DE LA MÉMOIRE - EXEMPLE

- Voici un exemple **problématique** (ne pas faire à la maison!!)

```
int fonction_mal_programmee(void) {
  int *entiers;
  int resultat;

  entiers = (int*)malloc(100 * sizeof(int));
  //autres instructions mais aucune qui libère la mémoire
  return resultat;
}
```

- Le pointeur *entiers* est une variable locale. Elle est détruite en quittant la fonction.
  → En quittant, une zone mémoire est réservée mais plus personne n'en connait l'existence et son adresse a été perdue!

17

FREE

- L'instruction **free** permet de libérer une zone mémoire réservée avec **malloc**, **calloc** ou **realloc**.
- **Prototype:**

  void free(void* ptr);

- Si ***ptr*** est NULL, free ne fait rien
- Si ***ptr*** est un pointeur qui pointe vers un espace qui n'a pas été réservé dynamiquement, son comportement est indéfini ☹.

18

FREE

- Correction de notre mauvais exemple:

```
int fonction_mieux_programmee(void) {
  int *entiers;
  int resultat;

  entiers = (int*)malloc(100 * sizeof(int));
  //autres instructions mais aucune qui libère la mémoire

  free(entiers);
  return resultat;
}
```


19

ALLOCATION DYNAMIQUE: CAS D'UTILISATIONS

Tableaux dynamiques, chaines de caractères et enregistrements.

20

TABLEAUX DYNAMIQUES 1D

- On alloue un espace pouvant contenir l'ensemble des valeurs à stocker dans le tableau. Ex.: pour un tableau de 30 **double**:

```
double *tab_doubles;
tab_doubles = (double*)malloc(30 * sizeof(double));
```

- Pour accéder aux cases du tableau, on utilise la notation traditionnelle []

```
tab_doubles[3] = 25.2;
//Note: ça fonctionne car:
// tab_doubles[3]  est equivalent à *(tab_doubles+3)
```


21

CHAINES DE CARACTÈRES

- Une chaine de caractères est un tableau à une dimension de caractères. On utilise donc le même procédé.

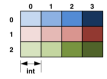
```
char *ma_chaine;
//On alloue l'espace pour la chaine de caractères – 20 caractères
ma_chaine = (char*)malloc(30 * sizeof(char));
//On stocke "Allo le monde!" dans la chaine
strncpy(ma_chaine, "Allo le monde!", 30);
//On affiche
printf("%s \n", ma_chaine);
```

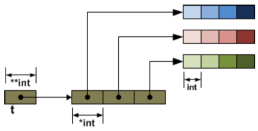

22

TABLEAUX 2D

- Il existe plusieurs façons d'allouer un tableau à deux (ou +) dimensions.
- La technique pour être en mesure d'utiliser la notation [][] (comme pour les tableaux statiques, consiste à:
  - Allouer **un** tableau 1D de lignes. Ce tableau contient des pointeurs vers des tableaux 1D de colonnes
  - Allouer **les** tableaux 1D de colonnes (un tableau par ligne)

```
int t[3][4];
```





23

TABLEAUX 2D

```
int **tab_2d;
int nb_lignes=5,
    nb_colonnes = 10;
//Allocation du tableau de lignes
tab_2d = (int**)malloc(nb_lignes * sizeof(int*));
assert(tab_2d!=NULL);
//Allocation des tableaux de colonnes
for(i=0; i<nb_lignes; i++){
   tab_2d[i]= (int*)malloc(nb_colonnes * sizeof(int));
   if(tab_2d[i]==NULL) { //Desallouer tout! }
}
```


24

EXAMEN FINAL DANS 3 SEMAINES!

25