Programmation impérative

ENSIIE

Semestre 1 — 2020–21

Allocations dynamiques

Comment retourner un tableau ?

Pas de possibilité de retourner un tableau statique

- Retourner une adresse (celle de la première case)

Problème : si un tableau est déclaré localement dans une fonction, il « disparaît » après l'appel à la fonction

Allocations dynamiques

Jusqu'ici : tout statique

Réserver zone mémoire pendant l'exécution ?

- Quelle taille ? en octets `sizeof(`*nom_type*`)`
- Réserver :

  `(`*nom_type*`) malloc(`*taille*`)`
  - retourne une adresse
- Libérer :

  `free(`*adresse*`)`
- Réserver n cases contiguës

  `(`*nom_type*`) calloc(`n`,` *taille*`)`

Allocations dynamiques

Initialisation :

- `malloc` : aucune (ce qui était là avant... ou pas)
- `calloc` : tout à 0

Allocations dynamiques très coûteuses en temps :

- recherche d'un bloc libre de bonne taille
  - assez grand
  - pas trop pour limiter "trous"
- éviter fragmentation quand libère mémoire
- compromis recherche efficace/utilisation mémoire optimale

`malloc/free` = mini "garbage collector"

Attention déréférencement !

Quand on accède au contenu d'un pointeur

- `*p`
- `p->x`

s'assurer que l'adresse est valide

Adresse toujours invalide

- `NULL`, définie dans `stdlib.h`

(attention, pas forcément 0 !)

Initialiser les pointeurs, quitte à utiliser `NULL`

- `p` valide $\Leftrightarrow$ `p != NULL`

Fuite mémoire

Il faut penser à désallouer les zones mémoires qui ne seront plus accessibles

```
int *t;
int *q;
int i;
t = (int *) malloc(42 * sizeof(int));
for (i = 0; i < 42; i = i + 1)
  t[i] = rand();
q = (int *) malloc(42 * sizeof(int));
for (i = 0; i < 42; i = i + 1)
  q[i] = t[41 - i];

t = q;
```

Fuite mémoire

Il faut penser à désallouer les zones mémoires qui ne seront plus accessibles

```
int *t;
int *q;
int i;
t = (int *) malloc(42 * sizeof(int));
for (i = 0; i < 42; i = i + 1)
  t[i] = rand();
q = (int *) malloc(42 * sizeof(int));
for (i = 0; i < 42; i = i + 1)
  q[i] = t[41 - i];
free(t);
t = q;
```