Langage C

ASR2 - Système

Troisième séance
(allocation dynamique)

17 avril 2013

Allocation dynamique

Pourquoi ?

- N'allouer la mémoire que lorsqu'on en a besoin
- La libérer quand on en a plus besoin
- N'allouer que la taille requise
- S'adapter dynamiquement à des changements de taille

Allocation dynamique

Pourquoi ?

- N'allouer la mémoire que lorsqu'on en a besoin
- La libérer quand on en a plus besoin
- N'allouer que la taille requise
- S'adapter dynamiquement à des changements de taille

Comment ?

- Allocation de base : `malloc` (∼équiv. C++ new)

Allocation dynamique

Pourquoi ?

- N'allouer la mémoire que lorsqu'on en a besoin
- La libérer quand on en a plus besoin
- N'allouer que la taille requise
- S'adapter dynamiquement à des changements de taille

Comment ?

- Allocation de base : `malloc` (∼équiv. C++ new)
- Changement de taille : `realloc`

Allocation dynamique

Pourquoi ?

- N'allouer la mémoire que lorsqu'on en a besoin
- La libérer quand on en a plus besoin
- N'allouer que la taille requise
- S'adapter dynamiquement à des changements de taille

Comment ?

- Allocation de base : `malloc` (∼équiv. C++ new)
- Changement de taille : `realloc`
- Libération : `free` (∼équiv. C++ delete)

Allocation dynamique

Pourquoi ?

- N'allouer la mémoire que lorsqu'on en a besoin
- La libérer quand on en a plus besoin
- N'allouer que la taille requise
- S'adapter dynamiquement à des changements de taille

Comment ?

- Allocation de base : `malloc` (∼équiv. C++ new)
- Changement de taille : `realloc`
- Libération : `free` (∼équiv. C++ delete)
- Allocation et initialisation à 0 : `calloc`

Allocation dynamique

Pourquoi ?

- N'allouer la mémoire que lorsqu'on en a besoin
- La libérer quand on en a plus besoin
- N'allouer que la taille requise
- S'adapter dynamiquement à des changements de taille

Comment ?

- Allocation de base : `malloc` (∼équiv. C++ new)
- Changement de taille : `realloc`
- Libération : `free` (∼équiv. C++ delete)
- Allocation et initialisation à 0 : `calloc`
- Autres fonctions spécialisées...

Exemples concrets

`char *str;` $\Longrightarrow$ à ce stade, `str` $\rightsquigarrow$?

Exemples concrets

`char *str;` $\Longrightarrow$ à ce stade, `str` $\rightsquigarrow$?

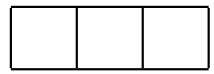
Trois fonctions clés (version simplifiée) :

- `str = malloc(3)`

Exemples concrets

`char *str;` $\Longrightarrow$ à ce stade, `str` $\rightsquigarrow$?

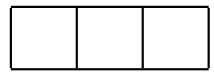
Trois fonctions clés (version simplifiée) :

- `str = malloc(3)` $\Longrightarrow$ str $\rightsquigarrow$ 

Exemples concrets

`char *str;` $\Longrightarrow$ à ce stade, `str` $\rightsquigarrow$?

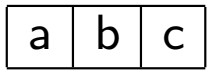
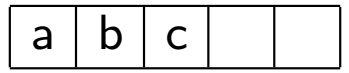
Trois fonctions clés (version simplifiée) :

- `str = malloc(3)` $\Longrightarrow$ str $\rightsquigarrow$ 
- `str = realloc(str, 5);`

Exemples concrets

`char *str;` $\Longrightarrow$ à ce stade, `str` $\rightsquigarrow$?

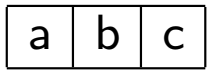
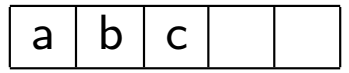
Trois fonctions clés (version simplifiée) :

- `str = malloc(3)` $\Longrightarrow$ str $\rightsquigarrow$ 
- `str = realloc(str, 5);` str $\rightsquigarrow$  $\Longrightarrow$ str $\rightsquigarrow$ 

Exemples concrets

`char *str;` $\Longrightarrow$ à ce stade, `str` $\rightsquigarrow$?

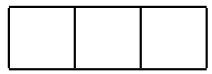
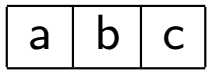
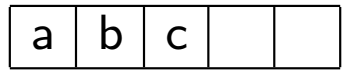
Trois fonctions clés (version simplifiée) :

- `str = malloc(3)` $\Longrightarrow$ str $\rightsquigarrow$ 
- `str = realloc(str, 5);` str $\rightsquigarrow$  $\Longrightarrow$ str $\rightsquigarrow$ 
- `free(str);`

Exemples concrets

`char *str;` $\Longrightarrow$ à ce stade, `str` $\rightsquigarrow$?

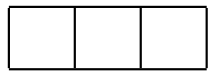
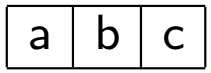
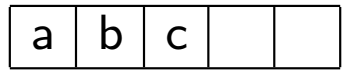
Trois fonctions clés (version simplifiée) :

- `str = malloc(3)` $\Longrightarrow$ str $\rightsquigarrow$ 
- `str = realloc(str, 5);` str $\rightsquigarrow$  $\Longrightarrow$ str $\rightsquigarrow$ 
- `free(str);` $\Longrightarrow$ str $\rightsquigarrow$?

Exemples concrets

`char *str;` $\Longrightarrow$ à ce stade, `str` $\rightsquigarrow$?

Trois fonctions clés (version simplifiée) :

- `str = malloc(3)` $\Longrightarrow$ `str` $\rightsquigarrow$ 
- `str = realloc(str, 5);` `str` $\rightsquigarrow$  $\Longrightarrow$ `str` $\rightsquigarrow$ 
- `free(str);` $\Longrightarrow$ `str` $\rightsquigarrow$?

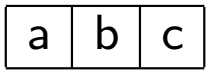
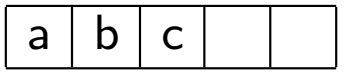
En fait (bonne pratique) :

- `str = malloc(3 * sizeof(char));`
- `str = realloc(str, 5 * sizeof(char));`

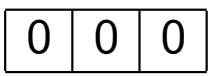
Exemples concrets

`char *str;` $\Longrightarrow$ à ce stade, `str` $\rightsquigarrow ?$

Trois fonctions clés (version simplifiée) :

- `str = malloc(3)` $\Longrightarrow$ `str` $\rightsquigarrow$ 
- `str = realloc(str, 5);` `str` $\rightsquigarrow$  $\Longrightarrow$ `str` $\rightsquigarrow$ 
- `free(str);` $\Longrightarrow$ `str` $\rightsquigarrow ?$

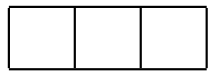
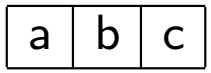
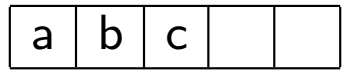
En fait (bonne pratique) :

- `str = malloc(3 * sizeof(char));`
- `str = realloc(str, 5 * sizeof(char));`
- `str = calloc(3, sizeof(char));` `str` $\rightsquigarrow ?$ $\Longrightarrow$ `str` $\rightsquigarrow$ 

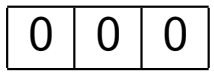
Exemples concrets

`char *str;` $\Longrightarrow$ à ce stade, `str` $\rightsquigarrow ?$

Trois fonctions clés (version simplifiée) :

- `str = malloc(3)` $\Longrightarrow$ `str` $\rightsquigarrow$ 
- `str = realloc(str, 5);` `str` $\rightsquigarrow$  $\Longrightarrow$ `str` $\rightsquigarrow$ 
- `free(str);` $\Longrightarrow$ `str` $\rightsquigarrow ?$

En fait (bonne pratique) :

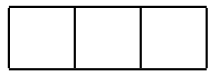
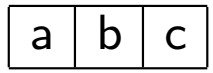
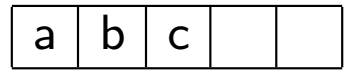
- `str = malloc(3 * sizeof(char));`
- `str = realloc(str, 5 * sizeof(char));`
- `str = calloc(3, sizeof(char));` `str` $\rightsquigarrow ?$ $\Longrightarrow$ `str` $\rightsquigarrow$ 

$\rightarrow$ Exercice 1 : récupérer `trieuse.c` sur la bibliothèque et remplir les trous.

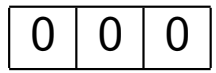
Exemples concrets

`char *str;` $\Longrightarrow$ à ce stade, `str` $\rightsquigarrow ?$

Trois fonctions clés (version simplifiée) :

- `str = malloc(3)` $\Longrightarrow$ `str` $\rightsquigarrow$ 
- `str = realloc(str, 5);` `str` $\rightsquigarrow$  $\Longrightarrow$ `str` $\rightsquigarrow$ 
- `free(str);` $\Longrightarrow$ `str` $\rightsquigarrow ?$

En fait (bonne pratique) :

- `str = malloc(3 * sizeof(char));`
- `str = realloc(str, 5 * sizeof(char));`
- `str = calloc(3, sizeof(char));` `str` $\rightsquigarrow ?$ $\Longrightarrow$ `str` $\rightsquigarrow$ 

$\rightarrow$ Exercice 1 : récupérer `trieuse.c` sur la bibliothèque et remplir les trous.

$\rightarrow$ Exercice 2 : adapter le code pour gérer des personnes plutôt que des entiers.

Trouver le bug

```
#include <stdlib.h>

void allouer(char *c, int nb)
{
    c = malloc(nb * sizeof(char));
}

int main()
{
    char *c;
    allouer(c,10);
    c[5]='a';
}
```

→ Que donne l'exécution de ce code ? Pourquoi ?

Trouver le bug

```
#include <stdlib.h>

void allouer(char *c, int nb)
{
    c = malloc(nb * sizeof(char));
}

int main()
{
    char *c;
    allouer(c,10);
    c[5]='a';
}
```

→ Que donne l'exécution de ce code ? Pourquoi ?

→ Que se passe-t-il au niveau des copies de paramètres ?

Trouver le bug

```
#include <stdlib.h>

void allouer(char *c, int nb)
{
    c = malloc(nb * sizeof(char));
}

int main()
{
    char *c;
    allouer(c,10);
    c[5]='a';
}
```

→ Que donne l'exécution de ce code ? Pourquoi ?

→ Que se passe-t-il au niveau des copies de paramètres ?

→ Comment corriger ce problème ?

Trouver le bug

```
#include <stdlib.h>

void allouer(char **c, int nb)
{
    *c = malloc(nb * sizeof(char));
}

int main()
{
    char *c;
    allouer(&c,10);
    c[5]='a';
}
```