# Mémoire : Le tas

# Contraintes de la pile

La **durée de vie** d'une variable locale dépend de la **stack frame** dans laquelle elle existe.

Pour **échanger des données** entre stack frame, on multiplie les **copies de valeurs**.

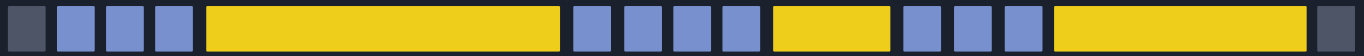
La **taille des tableaux** dans la pile est **fixe**.

La **taille de la pile** est relativement **limitée**.

**Il serait utile d'avoir une zone mémoire plus malléable que la pile !**

# Le tas : Définition

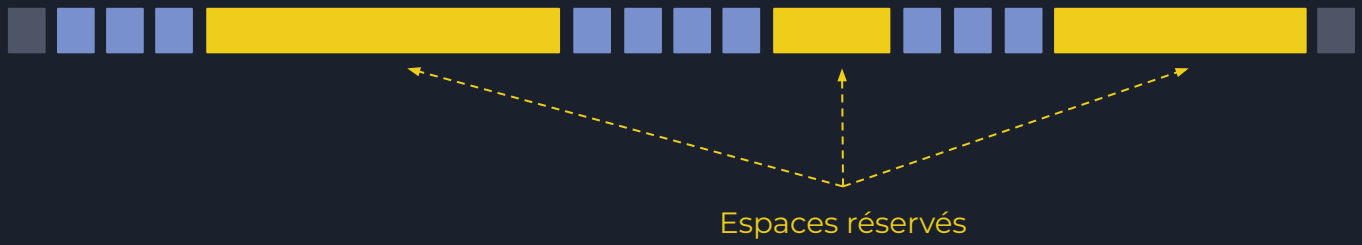
Le tas est un **segment de mémoire** qui permet de stocker des données **persistantes** et **indépendantes** de la **pile**.



# Le tas : Définition

En C / C++ : c'est au programmeur de **réserver** des **espaces mémoires** dans le tas.

**La durée de vie** d'un espace mémoire alloué dans le tas est **indéfinie**.



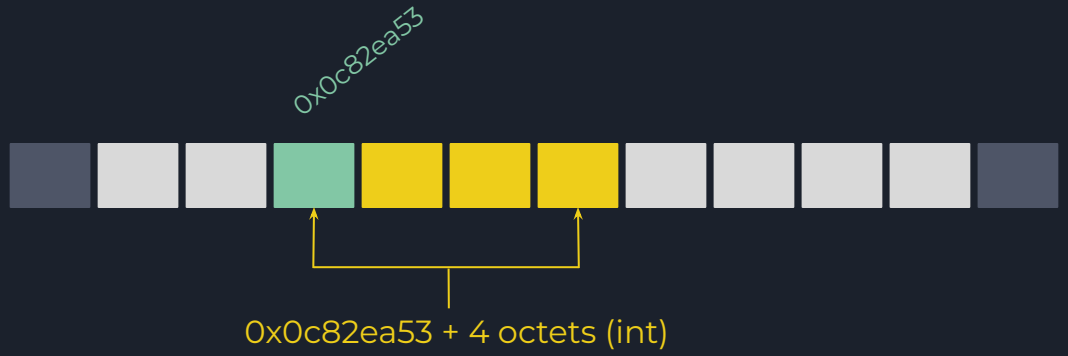
# Rappel

Pour comprendre le **fonctionnement du tas**, il est important de maîtriser les notions suivantes :

- Les tailles de variables.
- Les adresses mémoires.
- Les pointeurs.

# Rappel

**L'adresse mémoire** de début d'une variable et son **type** permettent de définir l'espace mémoire qu'une variable occupe dans la **RAM**.



# sizeof

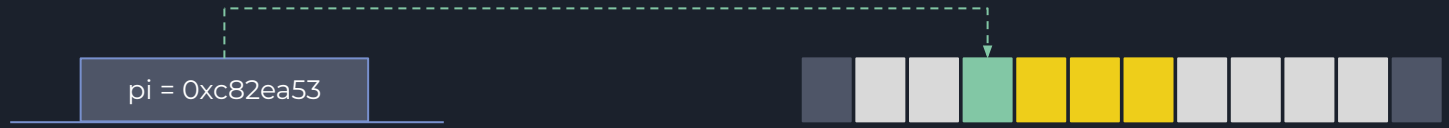
`sizeof()` : donne **la taille** en octets d'un type de valeur.

```
sizeof(char) //1
sizeof(int) //4
```

# new

`new` : demande au système d'exploitation **d'allouer un espace mémoire** dans le tas. Renvoie **l'adresse mémoire** de l'espace alloué.

```
int *pi = new int;
// pi vaut 0x0c82ea53
```



# Tas : pointeurs

Les **variables** dans le tas **n'ont pas d'identifiants**.

On les manipule en utilisant **l'adresse mémoire** à laquelle elles ont été allouées.

```
int *pi = new int;
*pi = 10;
```

# delete

`delete` : **libère (désalloue) un espace mémoire** du tas.

```
int *pi = new int;
delete pi;
```

# Fuites mémoires

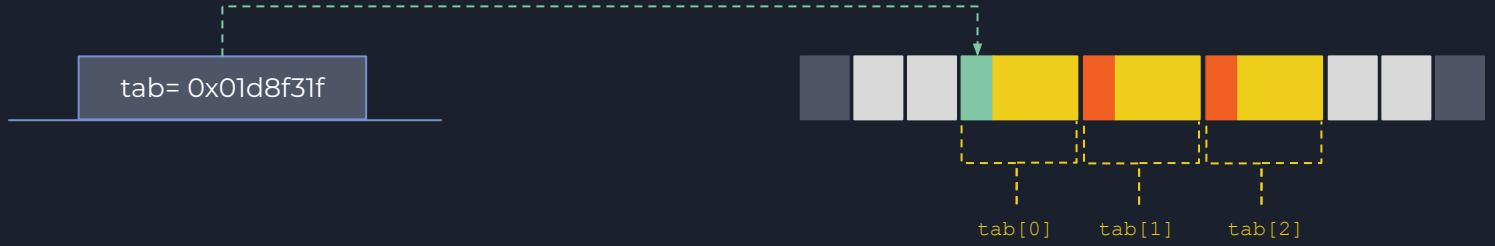
Un **espace mémoire** dans le tas qui n'est plus utilisé **doit être libéré manuellement**.

Oublier de libérer les espaces mémoires du tas encombre la RAM inutilement ! On appelle ce problème **une fuite mémoire**.

# Allocation de tableaux

Pour allouer un tableau dynamiquement :

```
int *tab = new int[3];
```



# Allocation de tableaux

Pour désallouer un tableau dynamiquement :

```
int *tab = new int[3];
delete[] tab; // Attention aux []!
```

# C++ : Opérateur ->

En C++, l'**opérateur ->** permet d'accéder à **une variable membre** depuis une adresse mémoire (structures et classes) :

```
struct Position { int x; int y; int z; };
Position* pos = new Position;
(*pos).x = 0;
pos->y = 1;
```