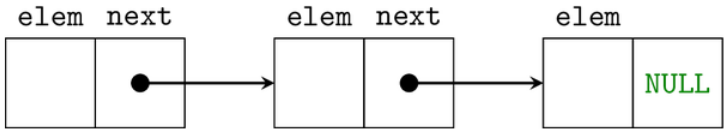
On implémente dans ce TP des listes chaînées en C. Ces listes chaînes contiendront uniquement des entiers. En voici la définition :

```
typedef struct list {
    struct list* next;
    int elem;
} list;
```

L'utilisation du `typedef` permet de raccourcir l'écriture, au lieu de devoir taper tout le temps `struct list` il suffira d'écrire `list`.

— Définir cette structure. Dans votre `main`, définir un exemple de liste contenant les trois valeurs $[1, 2, 3]$ et donc l'organisation suivante. Indice : partir de la fin, et définir les listes les unes après les autres.

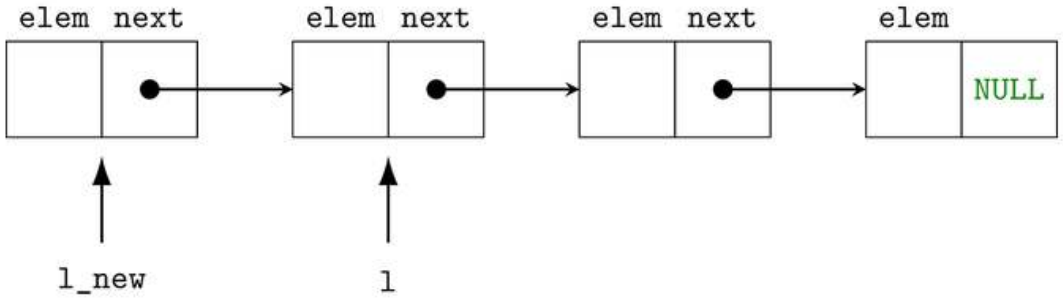


Ajout et création

— Écrire une fonction permettant d'ajouter un élément en tête d'une liste chaînée, avec le prototype suivant :

```
list* add(list* l, int e)
```

Cette fonction prend un pointeur `l` sur le début (premier nœud) d'une liste et renvoie un pointer `l_new` sur le nœud ajouté avant `l`, comme sur le dessin suivant :



— Écrire une fonction `list* range(int n)` qui renvoie une liste chaînée contenant les entiers de 0 à $n-1$ (comme en Python).

Parcours de liste

— Écrire une fonction `unsigned int longueur(list* l)` donnant la longueur d'une liste chaînée.

— Écrire une fonction `void print_list(list* l)` affichant les éléments d'une liste chaînée, au format suivant :

```
0 -> 1 -> 2 -> 3 -> 4 -> 5 ->
```

— Écrire une fonction `bool mem(list* l, int e)` pour savoir si un élément appartient à une liste chaînée (en important `stdbool.h` au préalable) :

Libérer la mémoire

Écrire une fonction `void free_list(list* l)` permettant de supprimer du tas (avec `free`) tous les nœuds d'une `list`.

Inverse

— Écrire une fonction `list* reverse(list* l)` pour inverser l'ordre des éléments d'une liste. On renverra un pointeur sur le premier élément de la liste inversé (qui est aussi le dernier élément de la liste `l` en argument).

Exercices en bonus

— Prendre une fonction quelconque du module `List` en OCaml, et essayer de l'implémenter en C sur ces listes simplement chaînées. Par exemple `List.exists`, `List.for_all`, etc.

