# TP 3

Dans le cadre de l'UE, vous êtes amenés à programmer en `C++` en utilisant les références pour réaliser vos passages de paramètres, les constructeurs, les destructeurs et les surcharges de l'opérateur d'affectation. Néanmoins, il sera important que vous sachiez toujours basculer dans un autre langage de programmation. Pour cela, il est important que vous établissiez votre raisonnement au niveau du pseudo-langage algorithmique et en utilisant les spécificités du langage utilisé seulement quand vous effectuez la mise en œuvre.

## 1 *Skip List*

Les *Skip lists* utilisent une stratégie du type pile ou face pour le maintien d'une structure de données qui optimise les opérations de recherche et d'insertion dans une séquence triée d'éléments. On recourt pour cela à une pièce de monnaie permettant d'obtenir *pile* avec la probabilité $p$ (et *face* avec la probabilité $(1-p)$).

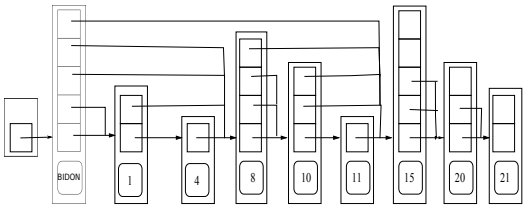


FIGURE 1 – Exemple de `Skip-List` avec implantation utilisant une cellule bidon. Cette `Skip-List` a 5 niveaux de chaînage. Chaque cellule possède ainsi un ou plusieurs pointeurs vers d'autres cellules (les suivantes dans chaque niveau), et chaque cellule est pointée par un ou plusieurs pointeurs. Voyez-vous quelles cellules sont dans chaque niveau ?

**Principe théorique des `Skip-List` :** Supposons que $x_1, x_2, ..., x_n$ soit une séquence triée d'éléments. Un premier échantillonnage de cet ensemble est réalisé en effectuant un jeu de pile ou face pour chaque élément, afin de décider si on le conserve comme échantillon représentatif ou non dans le niveau au-dessus. La sous-séquence sélectionnée est ensuite échantillonnée à son tour, en suivant le même principe, et ainsi de suite jusqu'à ce qu'il ne reste plus d'élément. C'est exactement ce principe que vous avez commencé à mettre en œuvre au TP précédent, sauf que vous n'aviez alors que deux niveaux.

Les différents niveaux de la structure chaînée modélisant la `Skip-List` correspondent aux sous-séquences obtenues par échantillonnage successif de la séquence à stocker. Ces différents niveaux s'articulent les uns les autres autour de leurs éléments communs. En pratique, les éléments sont stockés dans des Cellules, comme dans une Liste, et les éléments en commun ne sont pas dupliqués mais partagés par les différents niveaux, ce qui permet leur "articulation". Chaque niveau correspond à un chaînage différent des cellules avec des pointeurs différents (voir figure 1). Chaque cellule possède ainsi un pointeur vers la cellule suivante pour chaque niveau auquel elle appartient et chaque cellule est pointée par un ou plusieurs pointeurs. Lors de la recherche d'un élément, on commence à le chercher dans les éléments du niveau le plus haut. On localise ainsi le dernier élément inférieur à celui que l'on cherche dans ce niveau. La localisation relative dans un niveau est ensuite utilisée comme point de départ pour une localisation plus précise dans le niveau en

dessous, et ainsi de suite jusqu'à aboutir à la localisation dans le niveau de base. Le niveau le plus bas est celui qui contient tous les éléments de la séquence triée.

**Insertion d'un élément dans une `Skip-List` :** La description ci-dessus est faite sur une séquence triée déjà remplie, mais la structure de `Skip-List` peut également être maintenue dynamiquement en permettant l'insertion d'un nouvel élément : après une localisation efficace utilisant intelligemment tous les niveaux, l'insertion d'un élément s'effectue dans le niveau de base ; on effectue alors un jeu de pile ou face pour décider si l'élément devra également être inséré dans le niveau juste au-dessus, et ainsi de suite tant que l'on obtient *pile*. Quand un élément doit être supprimé, il est suffisant de le supprimer à tous les niveaux dans lesquels il apparaît.

**La structure de `Skip-List` peut être vue comme un arbre d'intervalles, dans lequel le nombre de fils d'un nœud n'est pas fixe. Un intervalle entre deux éléments consécutifs dans un niveau est ensuite découpé en sous-intervalle au niveau en dessous.**

Nous verrons en TD que le coût d'une insertion et d'une suppression se fait en temps logarithmique, par couplage avec l'opération de localisation.

- Comment s'effectuera la recherche de 8,5 dans la `Skip-List` donnée en exemple ?
- Le nombre de niveaux dans la `Skip-List` sera-t-il constant au fur et à mesure des insertions et des suppressions d'éléments ?
- Reprenez votre structure de données élaborée au TP précédent en augmentant le nombre de niveaux (cela signifie que dans une cellule vous aurez désormais un tableau de pointeurs, et dans un premier temps vous êtes autorisés à brider le nombre maximal de niveaux, pour pouvoir utiliser un tableau de taille fixe dans chaque cellule. Il faudra alors que vous stockiez le nombre de niveaux utilisés par la Cellule)
- Réfléchissez à la fonction de recherche d'un élément dans une `Skip-List`.
- Réfléchissez à la procédure d'insertion d'un élément dans une `Skip-List`.
- Proposez une stratégie de suppression d'un élément dans une `Skip-List`.

## 2 `Skip-List` modulaire

Complétez votre module `Skip-List` correspondant à une mise en œuvre efficace du type abstrait de donnée `Séquence triée`. Le type d'`Elément` utilisé est défini dans un module `Elément` (par exemple des int). L'interface du module `Skip-List` offrira des fonctionnalités de recherche, d'insertion et de suppression d'un élément (en plus des classiques opérations d'initialisation, affectation, testament, ...). Concernant les opérations d'initialisation, prévoyez un constructeur par défaut, en plus du constructeur à partir d'une `Liste` triée.

N'oubliez pas de tester vos fonctions et procédures au fur et à mesure, et d'utiliser une fonctionnalité d'affichage permettant d'avoir une vision de l'état interne de la `Skip-List`.

Exemple de programme utilisateur (pensez à vérifier que votre code résiste au passage de *valgrind*) :

```
SListe sissi;
SCell *p=NULL;
for(int i=0;i<10000;i++)
 { sissi.insertion(rand());}
p=sissi.recherche(15);
if(p!=nullptr)
   std::printf(''15 est dans la SListe \n'');
```