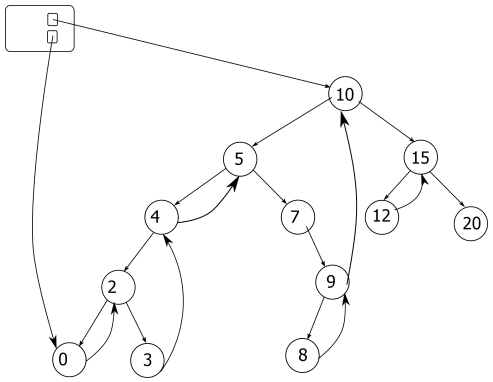
TP 7 : Arbres Cousus

Dans le cadre de l'UE, vous êtes amenés à programmer en **C++** en utilisant les références pour réaliser vos passages de paramètres ainsi que 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, en utilisant les spécificités du langage utilisé seulement quand vous effectuez la mise en œuvre.

1 `Arbres Binaires de Recherche Cousus` dans l'ordre infixé

Dans un `Arbre Binaire de Recherche` contenant `n` éléments quel est le nombre de pointeurs nuls ?

L'idée des arbres cousus consiste à utiliser les pointeurs non utilisés d'un `Arbre Binaire de Recherche` pour permettre le parcours des nœuds dans l'ordre infixé. Ainsi, il sera possible d'utiliser le "pointeur droit" d'un nœud qui n'a pas de sous-arbre droit pour pointer sur le nœud qui lui succède dans l'ordre infixé (voir figure). De la même manière, si l'arbre était doublement cousu, il serait possible d'utiliser le "pointeur gauche" d'un nœud qui n'a pas de sous-arbre gauche pour pointer sur le nœud qui le précède dans l'ordre infixé, mais nous nous contenterons ici de faire des arbres simplement cousus.



Nous vous proposons d'enrichir votre structure `Arbre Binaire de Recherche` avec une donnée membre permettant de savoir s'il est cousu ou non. Cela peut être fait sous la forme d'un pointeur qui sera nul si l'`Arbre Binaire de Recherche` n'est pas cousu, et qui pointera vers le plus petit nœud de l'arbre si l'arbre est cousu (voir figure). Concernant les nœuds d'un arbre cousu, le problème est de savoir à quel emploi est dédié leur "pointeur droit". Pour cela, vous ajouterez une donnée membre permettant de savoir si un nœud a un sous-arbre droit ou pas.

Codez la fonction transformant un `Arbre Binaire de Recherche` non cousu en `Arbre Binaire de Recherche` simplement cousu. L'idée consiste à réaliser un parcours en profondeur de l'arbre. Dans le chemin menant au nœud courant, il est important de conserver l'adresse du nœud le plus bas au dessous duquel on a bifurqué à gauche (on conserve la valeur *nullptr* sinon). Pour chaque nœud rencontré dont le pointeur droit est nul, on détourne l'utilisation de ce pointeur pour le faire pointer sur le nœud le plus bas dont on a stocké l'adresse à la descente, il s'agit en effet du nœud contenant l'élément suivant dans l'ordre infixé.

Pour cela, vous coderez une fonction interne récursive qui prendra un `Noeud n` en paramètre `Donnée-Résultat` de manière à pouvoir détourner l'utilisation de son "pointeur droit", dans le cas où celui-ci n'est pas utilisé pour pointer vers un sous-arbre. Cette fonction interne récursive prendra également en paramètre l'adresse du `Noeud` le plus bas rencontré sur le chemin menant à `n` et pour lequel le chemin s'est orienté sur la gauche.

Vous coderez ensuite une fonction permettant d'afficher les éléments stockés dans l'`Arbre Binaire de Recherche` dans l'ordre infixé. Ceci peut être réalisé avec un simple pointeur "de travail", sans utiliser de pile ni de procédure récursive : chaque fois que l'on s'oriente vers un noeud en suivant un pointeur de type fils droit, on descend aussitôt le plus loin possible sur la gauche, pour trouver le prochain élément à afficher. Lorsqu'on a affiché l'élément d'un noeud, on recommence si c'est possible l'opération d'aller vers la droite puis à gauche toute, sinon on utilise le nouveau lien. C'est ainsi que l'on trouve le prochain élément à afficher. Si le pointeur droit est nul, c'est que le parcours est achevé. Le successeur dans l'ordre infixé d'un Noeud qui n'a pas de fils droit est son plus jeune ancêtre pour lequel le parcours ne s'est encore effectué qu'à gauche (grâce ou pointeur de couture on s'évite ainsi la remontée récursive).

Modifiez la procédure d'insertion d'un élément dans un `Arbre Binaire de Recherche` de manière à ce qu'elle mette à jour l'aspect cousu de l'arbre suite à une insertion.

Vous avez tout terminé ? Les plus avancés d'entre vous pourrez commencer à coder les opérations de rotation vues en cours. Ces rotations sont utiles pour la mise en œuvre des AVL mais aussi d'autres types d'arbres équilibrés.