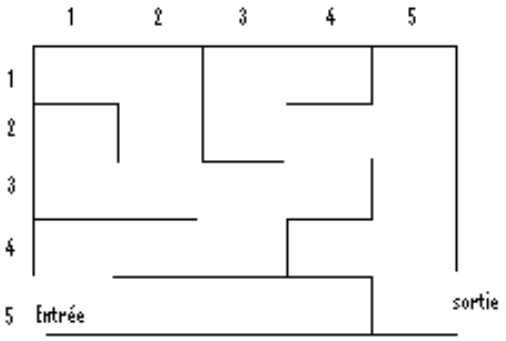
ARBRES

Exercice 1

Etant donné le labyrinthe de la figure ci-dessous. L'entrée de ce labyrinthe est donnée par la case (5,1) qui représente la 5ème ligne et la 1ère colonne de la matrice. La sortie est représentée par la case (5,5). On veut représenter ce labyrinthe par un **arbre binaire**. Les déplacements possibles dans ce labyrinthe sont : G : vers la gauche, D : vers la droite, H : vers le haut et B : vers le bas. Pour chaque nœud de l'arbre, on commence toujours par créer le fils gauche puis le fils droit en suivant les priorités suivantes : G : 1 (priorité élevée), D : 2, H : 3 et B : 4 (priorité faible).

Si à un moment donné on a le choix entre deux cases : aller vers la gauche ou aller vers le bas par exemple, on construit le fils gauche avec les coordonnées de la case de gauche (priorité égale à 1) et le fils droit par les coordonnées de la case d'en bas (priorité égale à 4).

1. Donner la déclaration de cet arbre
2. Construire l'arbre correspondant à l'ensemble des cases de ce labyrinthe.
3. Donner la profondeur de la case (2,4)
4. D'après les parcours suivants : RGD, GRD, DGR et GDR, quel est le parcours qui affiche le premier la case (5,5). De même, quel est le parcours qui affiche le dernier, la case (5,5). Afficher toutes les étiquettes de ces deux parcours.



Exercice 2

En définie la récursivité croisée ou indirecte par : une fonction A qui fait référence à une fonction B qui elle-même fait appel directement ou indirectement à A.

En se basant sur cette définition et on se référant à l'exemple donné par la figure 1 :

1. Donnez l'écriture mathématique des suites : U_{n+1} et V_{n+1} pour n≥0
2. Donnez le cas trivial (qui assure l'arrêt des appelles) de chaque suite
3. Ecrire en C les fonctions **<u>récursives</u>** int *U* (int n) et int *V* (int n) permettant de calculer le nième terme
4. Calculez U_5 et V_7

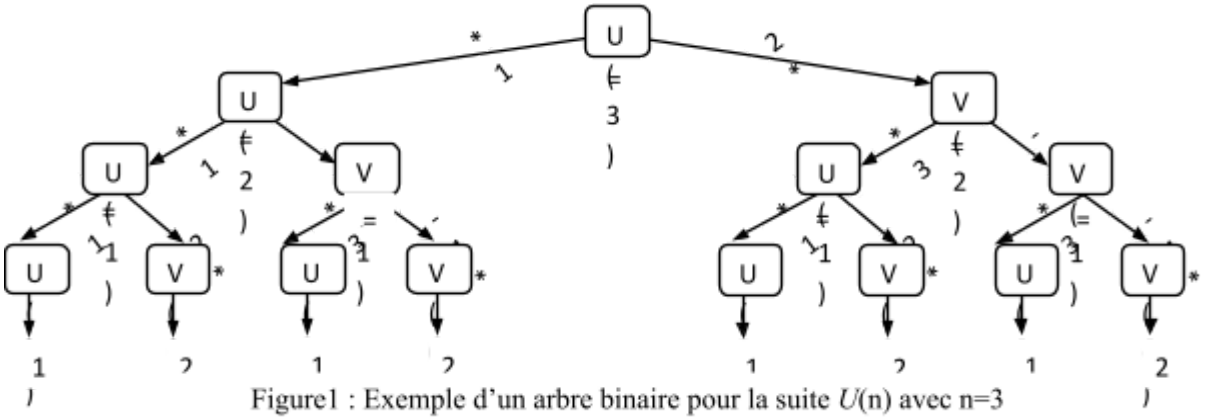


Figure1 : Exemple d'un arbre binaire pour la suite *U*(n) avec n=3

Exercice 3

Supposons qu'on a N objets différents. Chaque objet est codé par un nombre allant de 1 jusqu'à N. On veut générer toutes les combinaisons possibles en mettant ces objets les uns à côté des autres.

L'arbre de la figure 3 utilise 4 objets différents et montre la logique de construction des 6 permutations à partir du premier objet codé par le chiffre 1. Chaque chemin reliant la racine à une feuille donne une combinaison. Les combinaisons possibles données par cet arbre sont :

1234	1243	1324	1342	1423	1432

1. Donner le type de cet arbre.
2. Ecrire une action qui vérifie si un nœud donné est une feuille ou non.
3. En utilisant le parcours préfixe (Racine, Gauche, Droite dans le cas d'un arbre binaire), écrire l'action « Creer_Combine» qui crée toutes les combinaisons possibles (six combinaisons).
   Note : l'étiquette de chaque nœud doit être mise à jour par celle du nœud ascendant. A la fin de la procédure, chaque feuille doit contenir une combinaison parmi les six.
   Par exemple, pour le chemin le plus à gauche de cet arbre, on a :
   Le nœud de valeur 2 doit mettre à jour son étiquette par : 2 + (1 * 10) = 12
   Le nœud de valeur 3 doit mettre à jour son étiquette par: 3 + (12 * 10) = 123
   Le nœud de valeur 4 doit mettre à jour son étiquette par : 4 + (123 * 10) = 1234
4. Ecrire l'action récursive « Affiche_Combine» qui affiche les combinaisons possibles en utilisant l'action de la question 2.
5. Dans le programme principal, donner l'instruction qui appelle l'action « Creer_Combine».
6. Ecrire une action récursive qui recherche une valeur A dans l'arbre et change A par B. Les paramètres de cette action sont : la racine de l'arbre et les valeurs A et B.
7. En changeant le champ valeur de la racine par la valeur 2 (puis par 3 et 4) et en écrasant la valeur 2 par la valeur 1 dans les autres nœuds, on peut générer d'autres combinaisons (voir la figure 4).
   En suivant ce principe, écrire une action qui affiche les 24 combinaisons possibles du nombre 1234 (voir le tableau ci-dessous).
   Note : à la mis à jour d'un nouvel arbre, il est très utile d'utiliser le dernier.

1234	1243	1324	1342	1423	1432
2134	2143	2314	2341	2413	2431
3124	3142	3214	3241	3412	3421
4123	4132	4213	4231	4312	4321

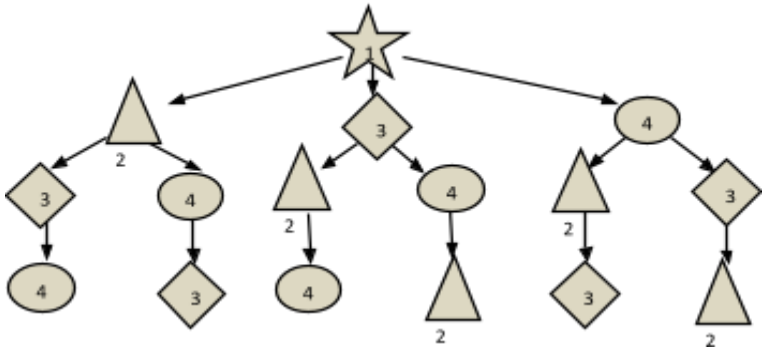


Figure 3 : arbre avec 4 objets différents. Chaque chemin reliant la racine à une feuille donne une combinaison. Six combinaisons possibles pour cette configuration.

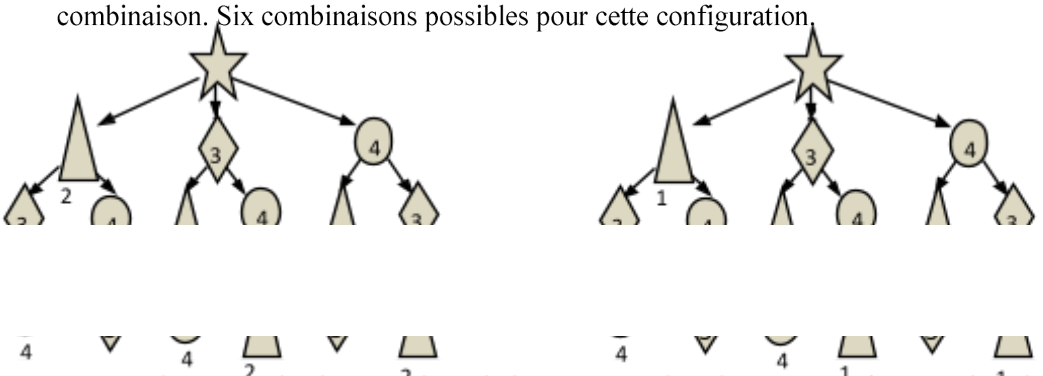


Figure 4 : En changeant le champ valeur de la racine par la valeur 2 et en écrasant la valeur 2 par la valeur 1 dans les autres nœuds on peut générer d'autres combinaisons.

Exercice 4

Un nombre est **narcissique** d'ordre *p* s'il est égal à la somme de ses *p* chiffres élevés à la puissance *p*.

Exemple : 153 = $1^3+5^3+3^3$, 1634= $1^4+6^4+3^4+4^4$, 54748=$5^5+4^5+7^5+4^5+8^5$…

On veut vérifier si un nombre est narcissique ou non en utilisant un arbre binaire où chaque feuille contient un chiffre (voir la figure 1).

1. Définissez le type de cet arbre.
- Créez une **fonction** qui initialise la racine d'un arbre binaire. Le seul paramètre de cette fonction est l'entier positif *N*.
- Créez une **procédure récursive** qui complète l'arbre binaire. Le seul paramètre de cette procédure est la racine de la question précédente.
- Ecrivez une **action récursive** qui retourne le nombre de feuille de cet arbre.

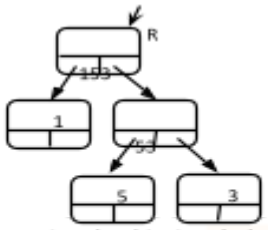


Figure 1 : arbre binaire où chaque feuille contient un chiffre

6. En appelant la procédure F (voir page 2) trois fois de suite pour l'arbre de la figure 1, donnez en une phrase le rôle de cette action. Complétez l'en-tête de cette action en justifiant votre choix.
7. Dans le programme principal vérifiez si les éléments de ce tableau sont narcissiques ou non :

3	407	55	9474

On veut étendre la définition d'un nombre narcissique pour chercher les couples (N_i, N_j) $i \neq j$ tel que la somme des n_1 chiffres chacun élevé à la puissance p=3 de N_i= N_j et la somme des n_2 chiffres chacun élevé à la puissance p=3 de N_j= N_i.

Exemple : La somme des n_1 chiffres chacun élevé à la puissance p=3 de **919** donne : $\mathbf{9^3 + 1^3 + 9^3 = 1459}$ et la somme des n_2 chiffres chacun élevé à la puissance p=3 de **1459** donne : $\mathbf{1^3 + 4^3 + 5^3 + 9^3 = 919}$. Conclusion : $\mathbf{919 = 1^3 + 4^3 + 5^3 + 9^3}$ et $\mathbf{1459 = 9^3 + 1^3 + 9^3}$.

De même pour le couple (136, 244) : $\mathbf{136= 2^3 + 4^3 + 4^3}$ et $\mathbf{244 = 1^3 + 3^3 + 6^3}$.

8. Etant donné la définition de la liste ci-contre. Créez une liste de *N* entier strictement positif où le champ racine de chaque élément est initialisé à nil (voir l'exemple de la figure 2).

9. En utilisant les actions crées précédemment, trouvez et gardez uniquement les couples (N_i, N_j) $i \neq j$ vérifiant la définition donnée ci-dessus.

Note :

- Si pour un nombre N_i il n'existe aucun nombre N_j tel que le couple (N_i, N_j) $i \neq j$ vérifie la définition ci-dessus, on supprime l'arbre correspond à N_i ainsi que l'élément N_i de la liste.
- Si le couple (N_i, N_j) $i \neq j$ vérifie la définition, mettez à jour l'élément N_j pour qu'il soit le suivant directe de N_i. La liste finale contiendra tous les couples vérifiant la définition donnée ci-dessus (voir l'exemple de la figure 3).
- Si le couple (N_i, N_j) $i \neq j$ vérifie la définition, on ne cherche pas si (N_i, N_k) $i \neq k$ ni (N_j, N_k) $j \neq k$ vérifie ou non la définition.

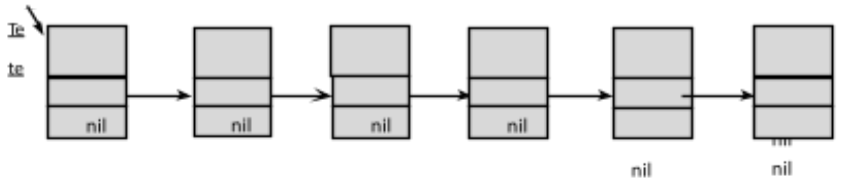


Figure 2: Etat initial

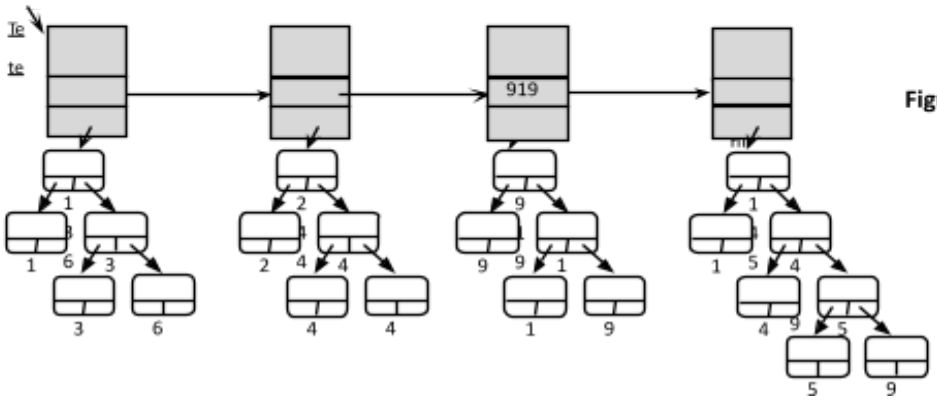


Figure 3: Etat final

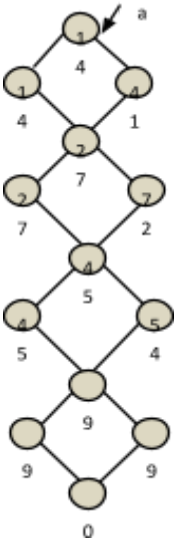
```
Procedure F ( ?????/ A : Arbre)  // COMPLETEZ L'EN-TETE DE CETTE ACTION ET JUSTIFIEZ
Debut
  B : booléen ;
  Prd : Arbre ;

  B Faux ;   Prdnil ;
  TQ (B=Faux)
    Faire
      Si (A^.fg=nil  et A^.fd=nil) alors BVrai ;
                                   sinon
                                     Si (A^.fg ≠ nil) alors PrdA ;  A A^.fg ;
                                                      sinon PrdA ;  A A^.fd ;
                                     Fsi ;
      Fsi ;
    Fait ;
  Si (Prd = nil) alors Liberer A ;  Anil ;
                 sinon
                   Si (Prd^.fg=A) alors   Prd^.fgnil ;  Liberer A ;
                                  sinon  Prd^.fdnil ;  Liberer A ;
                   Fsi ;
    Fsi ;
Fin
```

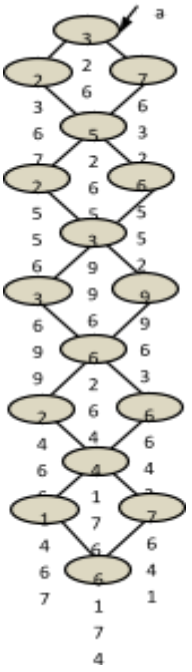
Exercice 5

Le procédé de **Kaprekar** consiste à ordonner les chiffres d'un nombre par ordre croissant (Min---->Max) puis par ordre décroissant (Max---->Min) puis effectuer leur soustraction (D = Max – Min).

La différence (D) est soumise à nouveau à ce même procédé.

Par exemple (voir les figures ci-contre) :





Le nombre 14 devient 27 (=41-14), puis 45 (=72-27) puis 9 (=54-45) et le procédé prend fin à 0 (=9-9).

Le cycle de Kaprekar de 14 est: [14, 27, 45, 9, 0]

La longueur du cycle est égale à 5

Le nombre final est 0

Le nombre 3267 devient 5265 (=7632-2367), puis 3996 (=6552-2556) puis 6264 (=9963-3699), puis 4176 (=6642-2466) et le procédé prend fin à 6174 (=7641-1467) car le nombre 6174 à les mêmes chiffres que le nombre 4176.

Le cycle de Kaprekar de 3267 est: [3267, 5265, 3996, 6264, 4176, 6174]

La longueur du cycle est égale à 6

Le nombre final est 6174

Questions :

1. Ecrire une action qui **retourne une liste chaînée** représentant les chiffres d'un nombre entier supérieur à zéro, **et le nombre d'élément** de cette liste.
2. Ecrire une action qui trie une liste par **ordre croissant** (Min---->Max). Il est recommandé de permuter uniquement les valeurs et non pas les éléments.
3. Ecrire une action qui trie une liste par **ordre décroissant** (Max---->Min). Il est recommandé de permuter uniquement les valeurs et non pas les éléments.
4. Ecrire une **fonction récursive** qui calcul un nombre à partir d'une liste chaînée : t1 et le nombre d'élément de cette liste.

   Fonction calcul_nb_liste(t1 : liste, nb_elt : entier) :entier
5. Ecrire une **action récursive** qui cherche une valeur dans un arbre binaire.
6. Ecrire une action qui **affiche les éléments d'un arbre binaire** niveau par niveau.
7. Ecrire une action qui **affiche le cycle de Kaprekar**.
8. Ecrire une action qui **supprime une liste**.
9. Ecrire l'algorithme suivant :

Algorithme

1. Choisir un nombre N
2. Former le Max (voir les actions de la question : 1, 3 et 4)
3. Former le Min (voir les actions de la question 1,2 et 4)
4. Calculer la différence D = Max – Min
5. Si D est un nombre appartenant à l'arbre (voir l'action de la question 5)

   alors aller à 6,

   sinon N prend la valeur D et aller en 2.
6. Afficher l'arbre (voir l'action de la question 6)

7. Afficher la longueur du cycle de Kaprekar
8. Afficher le cycle de Kaprekar (voir l'action de la question 7)
9. Fin

Remarque :

- x^n est donnée par : fonction puis (E/ x : entier, E/ n : entier) : entier
- Toute nouvelle action doit être prédéfinie avant son utilisation

Exercice 6

Étant donné deux arbres binaires de recherche (ABR) R1 et R2. A partir de ces deux arbres, on veut construire un troisième : R3. La procédure à suivre est la suivante :
On supprime la racine du premier et on insère dans le troisième puis on supprime du deuxième et on insère dans le troisième. Répéter ces étape jusqu'à épuiser tous les nœuds des deux arbres.

Note :

- On peut utiliser les actions suivantes vues en cours :
  - L'action remplace () pour supprimer un nœud ayant deux fils.
    Procedure remplace(ES/ max : arbre, E/ elt_sup : arbre, E/ prd_max : arbre)
  - L'action rechercher_iterative() pour chercher une valeur dans un ABR.
    Fonction rechercher_iterative(E/ A : arbre, ES/ prd : arbre, E/ x : entier) : booleen
- Les deux ABR n'ont aucune étiquette en commun.

Exercice 7

On considère les expressions arithmétiques utilisant les opérateurs suivants : +, -, *, /. Ces expressions peuvent être représentées par des arbres binaires dont les nœuds internes sont étiquetés par l'un des quatre opérateurs tandis que les feuilles sont étiquetées par des entiers. Par exemple, l'arbre représenté sur la figure 1 représente l'expression arithmétique (2*5)-(15+(10/5))+13.

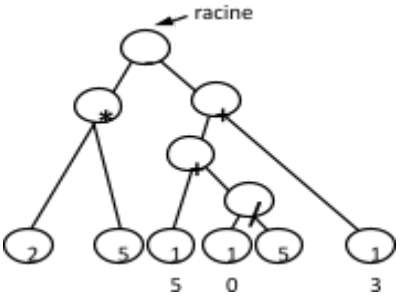


Figure1 : représentation de l'expression (2*5)-(15 + (10/5)) +13 par un arbre binaire

1. Donner la déclaration de cet arbre.
2. Dessiner l'arbre binaire pour l'expression suivante : X= (24/6)+19*((11+4)-10)/5).
3. En utilisant la fonction evaluer (T), évaluer l'expression X **en donnant le détail des appels**.

```
Fonction evaluer (E/ T: arbre) : réel
Début
  Cas où T^.info vaut
       '+' : retourner (evaluer(T^.Gauche) +  evaluer(T^.Droit))
       '-' : retourner (evaluer(T^.Gauche)  -  evaluer(T^.Droit))
```

```
        '*' : retourner (evaluer(T^.Gauche)  *  evaluer(T^.Droit))
        '/' : retourner (evaluer(T^.Gauche)  /  evaluer(T^.Droit))
    Autre : retourner (Trans_Ch_nb(T^.info))
  Fin cas
Fin
// Trans_Ch_nb ( ) est une fonction qui transforme une chaîne en un nombre. Exemple : Trans_Ch_nb (''1421'') donne  1421
```

4. Transformer la fonction evaluer(E/ T : arbre) afin de prendre en compte le nœud le plus à droite puis le nœud le plus à gauche. Evaluer l'expression X en tenant compte de cette transformation.