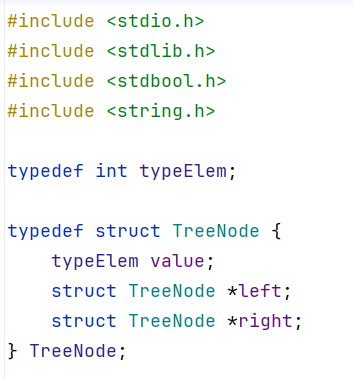
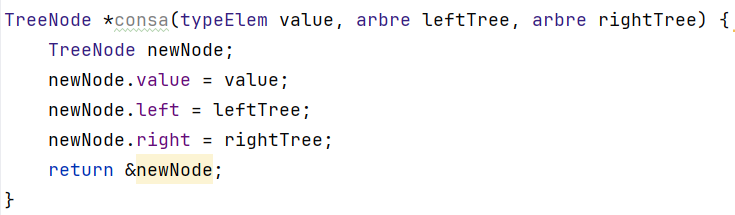


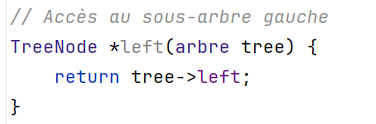
Rapport de laboratoire sur PA3

|  |  |
| --- | --- |
| **Projet** | TD5、TD6 |
| **Classe** | BD212 |
| **Nom** | Elie(金卓远)、Henri(叶秉文) |
| **ID** | 1211024054、1210514066 |

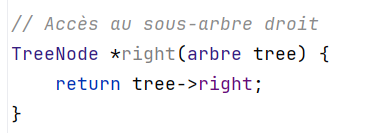
**Professeur MinRui**

### 2023年 12 月 18 日

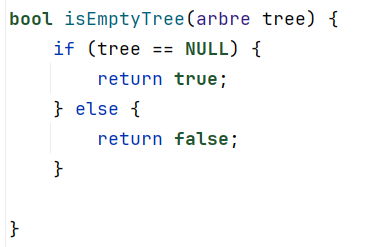
1. Préparation
   1. Définir une structure TreeNode comme un nœud de l'arbre
   2. Création d’un nouveau nœud
   3. Accès au sous-arbre gauche



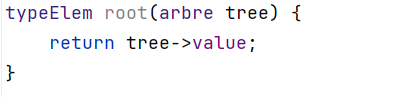
* 1. Accès au sous-arbre droit



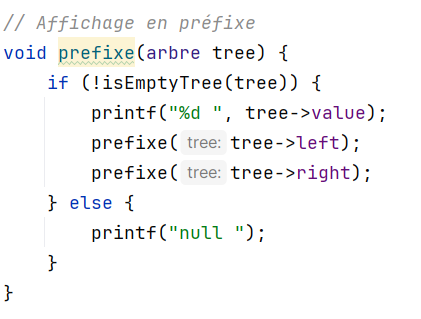
* 1. Déterminer si l'arbre est vide



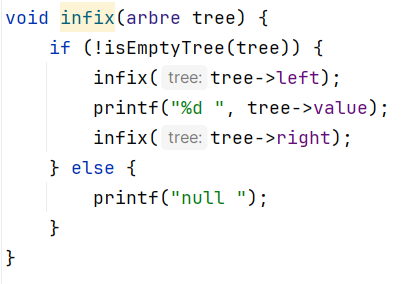
* 1. Donnez la racine de l’arbre



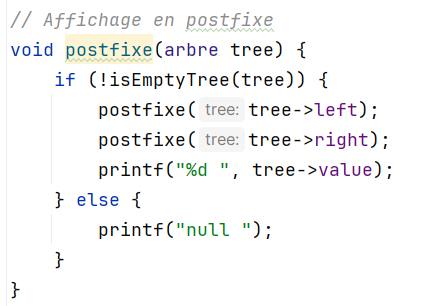
* 1. Afficher l’arbre
     1. Préfixe

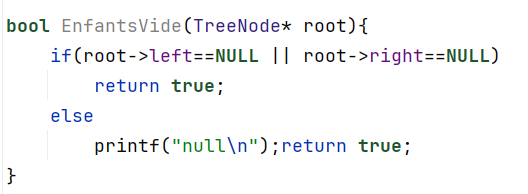


* + 1. Infixe



* + 1. Postfixe



* 1. Afficher null pour les “enfants vides” 

Les autres fonctions：

Création d'un arbre binaire à partir d'un tableau

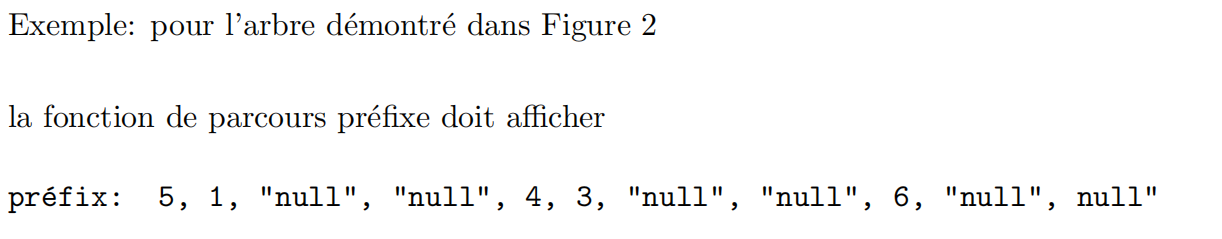


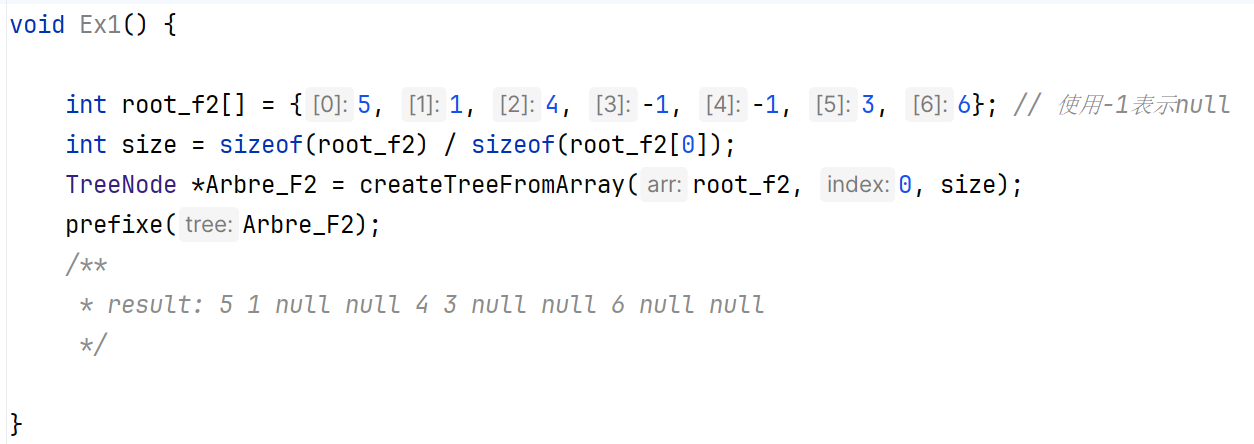
Mise en place d'une file d'attente



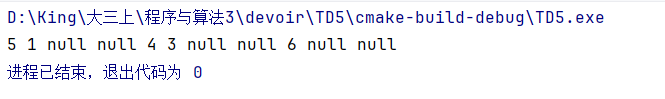
Parcours BFS（Parcours en largeur）







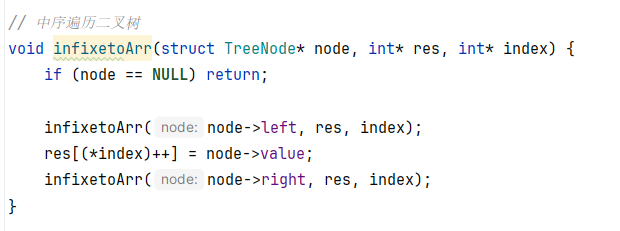
Résulta：



1. Exercice2

Parce que l’infixe d’un arbre binaire trié correct est ascendant. Par conséquent, nous utilisons cette méthode pour vérifier que l'arbre de tri binaire est correct.

La fonction du code suivant est de parcourir un arbre binaire dans l'infixe et de stocker le résultat dans le tableau « res »



le pseudo-code :

fonction isValidBST(root Arbre) :vrai ou faux

début

si root=NULL alors

retourner vrai ;

//Pour obtenir le nombre de nœuds dans l'arbre

nombreDeNoeud <- 0 ;

//créer un tableau pour mettre le résultat d’infixe

A<-tableau\_vide ;

//Parcours en infixe et stockage des résultats dans un tableau

infixetoArr(root,A,nombreDeNoeude) ;

//Vérifier si le tableau est trié(si précédent plus suivant alors faux)

Pour i<- 1 au nombreDeNoeud faire

Si A[i]<=A[i-1] alors

Libérer A ;

Retourner faux ;

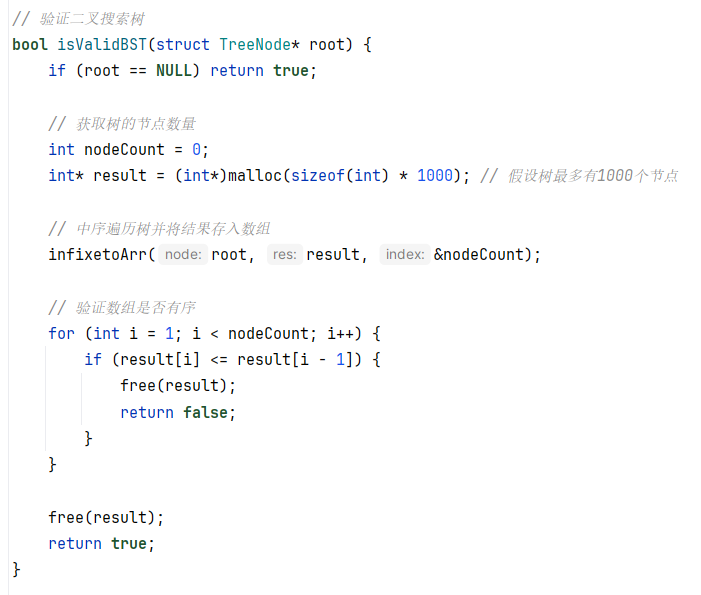
Fin si

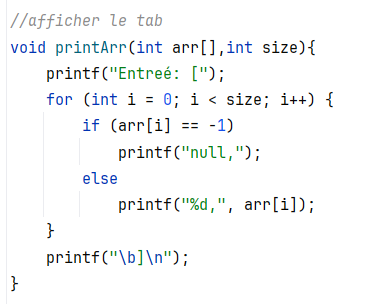
Fin pour

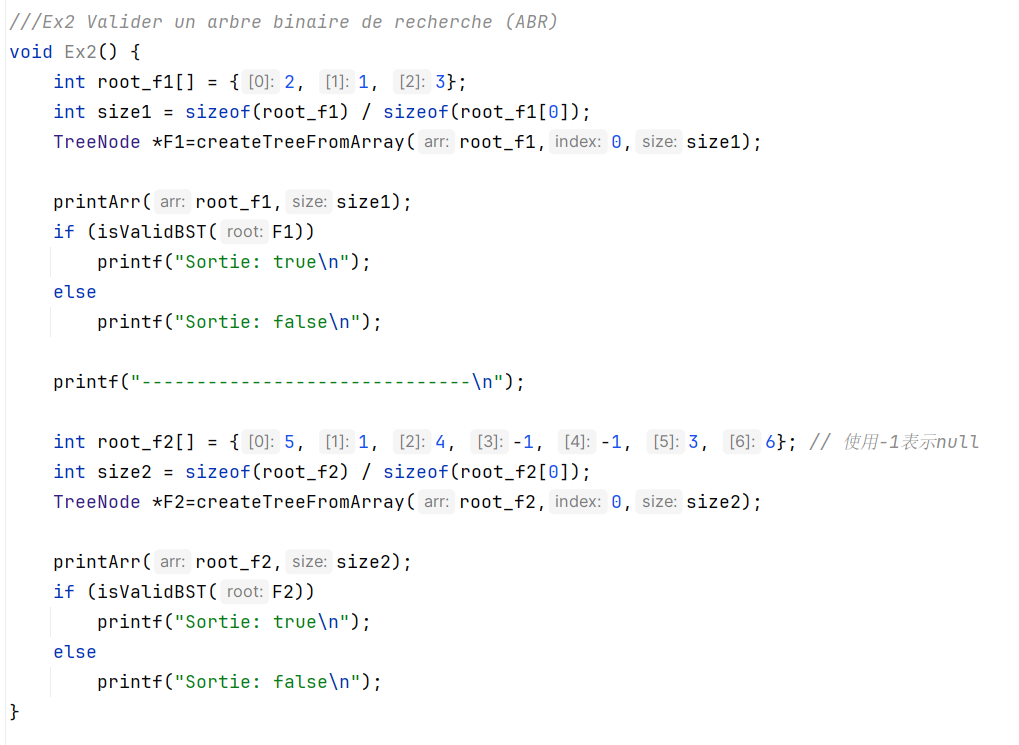
Libérer A ;

Retourner vrai ;

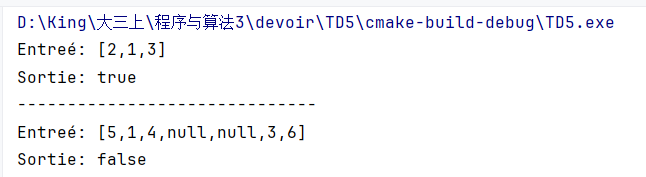
Fin







Résultat :



1. Exercice3

Récupérer l’arbre binaire de recherche

D'après la question, seuls deux nœuds ont été échangés de manière incorrecte. J'ai utilisé le parcours de la racine centrale pour trouver les deux nœuds qui ont été échangés de manière incorrecte et je les ai échangés à nouveau.

le pseudo-code :

Fonction recoverTree(root Arbre) :vide

Début

// Créer un tableau de nœuds pour enregistrer deux échanges d'erreurs

Err[2]<- Tableau\_Vide

Si Arbre est vide alors

Retourner ;

Fin si

//Pour enregistrer le nœud précédent

Pre <- Nœud\_Vide

//définir une pile

Stack {Data Tableau,top Entier};

//initialisation de la pile

Top de Stack <- -1;

Tant que stack.top !=-1 et root != NULL faire

Si root != NULL alors

// Mettre le nœud courant sur la pile et se déplacer vers le sous-arbre de gauche

Data<- root ;

Left(root) ;

Sinon

// Sauter le nœud supérieur de la pile et vérifier s'il y a des erreurs

Root<- Data[top] ;

Si pre != NU LL et value de pre > value de root alors

// Enregistrer le premier nœud d'erreur

Err[0]<- le premier nœud d'erreur

// Enregistrer le deuxième nœud d'erreur

Err[1]<- le deuxième nœud d'erreur

Fin si

// Mise à jour du nœud précédent au nœud actuel

Pre<-root ;

// Passer au sous-arbre de droite

Right(root) ;

Fin si

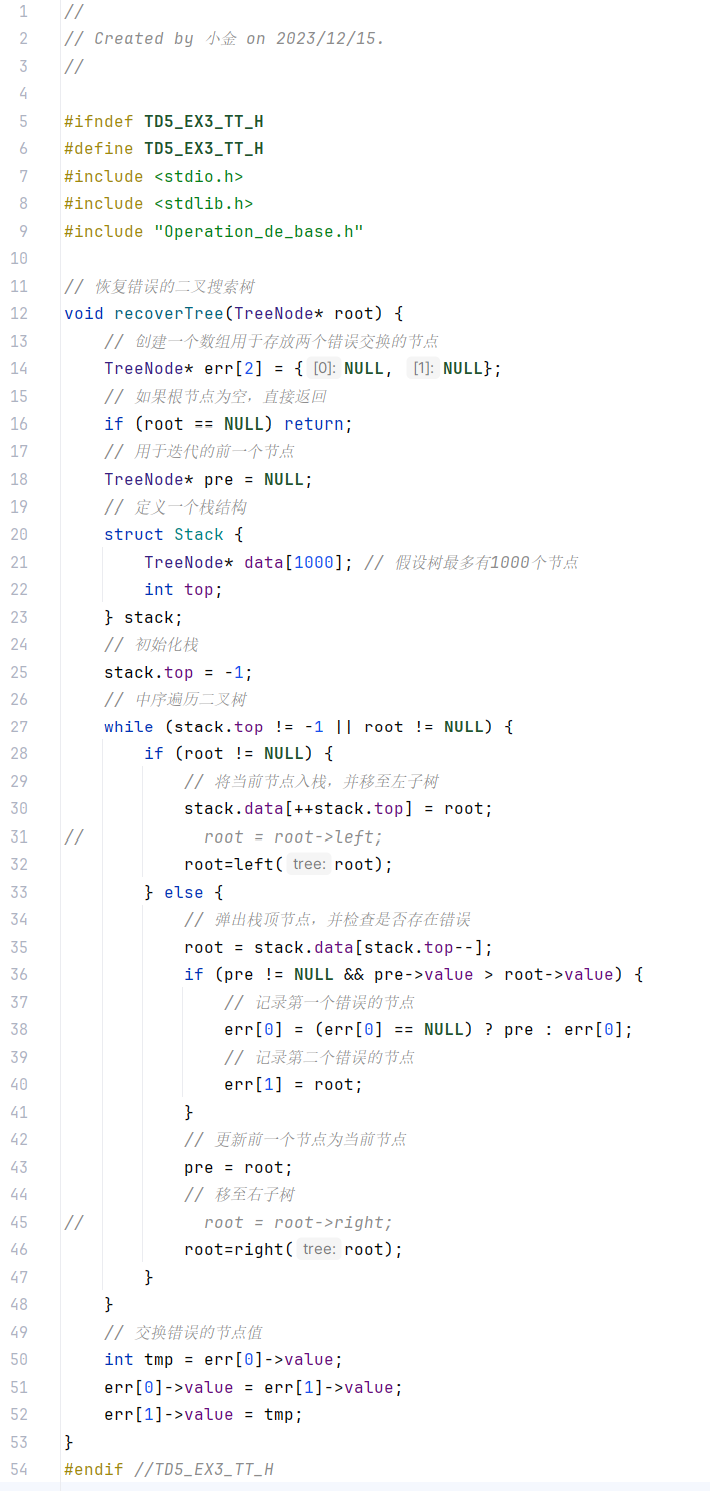
//Échanger le mauvais nœud

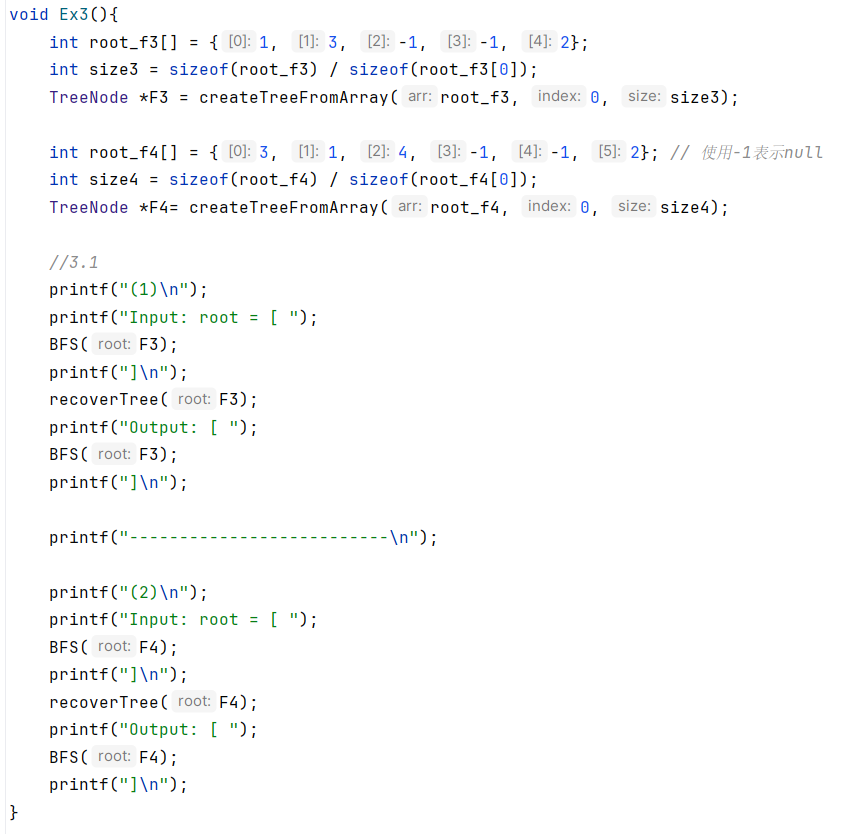
Tmp<-Err[0] ;

Err[0]<-Err[1] ;

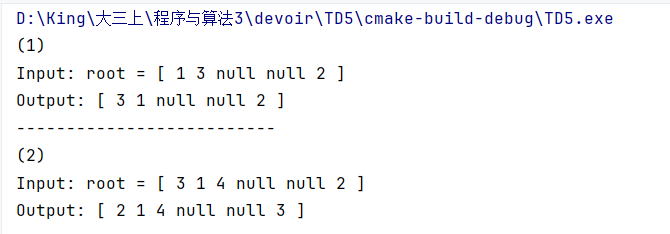
Err[1]<-tmp ;

Fin





Résultat :



1. Exercice4

Construction d’un arbre binaire à partir des parcours en préfixe et en infixe

le pseudo-code :

Fonction buildTree\_prefixe\_infixe(prefixe Tableau,infixe Tableau) :Arbre

Début

Si taille de préfixe est 0 alors

Retourner Arbre vide ;

Fin si

// Créer le nœud racine et le placer sur la pile

Root<-le premier élément de préfixe tableau

Ajouter root au stack ;

// Trouver la position du nœud actuel dans un tableau de traversée d'ordre moyen

infixeIndex<-0 ;

//Parcours du tableau en préfix (à partir du deuxième nœud)

Pour i<-1 à taille de préfixe tableau faire

Value\_prefixe<-prefixe[i]

Node <- le dernière élément de stack

// Si la valeur du nœud actuel n'est pas égale à la valeur du tableau dinfixe

Si value\_node != infixe[infixeIndex] alors

// Créer le nœud enfant de gauche et le placer sur la pile

Node.left<-value\_prefixe ;

Ajouter Node.left au stack ;

Sinon

//Si la valeur du nœud actuel est égale à la valeur du tableau d’infixe

// Retirer le nœud supérieur de la pile à tour de rôle jusqu'à ce que la pile soit vide ou que la valeur du nœud supérieur de la pile ne soit pas égale à la valeur du tableau de traversée d'infixe

Tant que stack n’est pas vide et l’élément de stack =infixe[infixeIndex] faire

Node<-le dernière de stack ;( Retirer l'élément de la pile)

infixeIndex<-infixeIndex+1 ;

//Créer un nœud enfant droit et le placer sur la pile

Node.right<-value\_préfixe;

Ajouter node.right au stack ;

Fin si

Fin pour

Retourner root ;

Fin

