Exercice 2 : 05 pts (0,5+ 3,5+ 01), 30 mn

On se propose de réaliser une calculatrice évaluant les expressions en notation postfixe. L'alphabet utilisé est le suivant : A = {0, . . . , 9,+,−, *, /} (l'opérateur – est ici binaire). Pour un opérateur ***P*** et les opérandes $Op_1, \ldots, Op_n$, l'expression, en notation postfixe, associée à ***P*** sera : $Op_1, \ldots, Op_n$ ***P***. Ainsi, la notation postfixe de l'expression (3 *5) + 7 + (4 * 2) sera : 3 5 *7+ 4 2 * +. On suppose que l'expression est valide et que les nombres utilisés dans l'expression sont des entiers. De plus, l'expression est donnée sous forme de chaînes de caractères terminée par un **#**. Par exemple (2 ∗ 5) + 6 + (4 ∗ 2) sera donnée par la chaîne 35 ∗ 7 + 42 ∗ + **#**.

1. Donner l'arbre binaire de l'expression : (3 *5) + 7 + (4 * 2).
2. Écrire un algorithme qui évalue une expression postfixe à l'aide d'une pile d'entiers.
3. Donner la trace d'exécution pour l'expression (2*5) +3.
   Remarque : les parenthèses ne sont pas prises en considération.

Exercice 3 : 08 pts (0,5+0,5+01+0,5+02+1,5+02), 70 mn

1. Rappeler les propriétés des arbres binaires de recherche.
2. Définir une structure ABR permettant de coder un nœud d'ABR contenant un entier (en utilisant notamment des pointeurs).
3. Écrivez une fonction *Cree_arbre* qui prend en argument deux entiers **X** et **Y** et renvoie un ABR de deux nœuds contenant ces valeurs, un parmi eux est un fils gauche.
4. Quel type de parcours donne l'arbre trié par ordre croissant ?
5. Écrire une fonction *Affiche_croissant* permettant d'afficher les valeurs des nœuds d'un ABR de manière croissante.
6. Écrire une fonction *Ajoute_entier* permettant d'ajouter un entier dans l'ABR (ce sera une nouvelle feuille bien placée dans l'arbre).
7. Écrire la fonction *Adresse_noeud* qui prend en argument un entier **X** et un arbre **A** et qui renvoie l'adresse d'un nœud de l'arbre contenant cet entier (ou NULL si cet entier n'est pas dans l'arbre).

Corrigé Type

Exercice 1

0,5 1. *« c'est une notion importante de la programmation qui permet de régler des problèmes extrêmement complexes avec seulement quelques lignes. C'est cependant une méthode avec laquelle il est facile de se perdre et d'avoir des résultats imprévisibles ou erronés.»*

Ici nous décrivons …

■ La récursivité	☐ La redondance	☐ Le parallélisme	☐ Autre

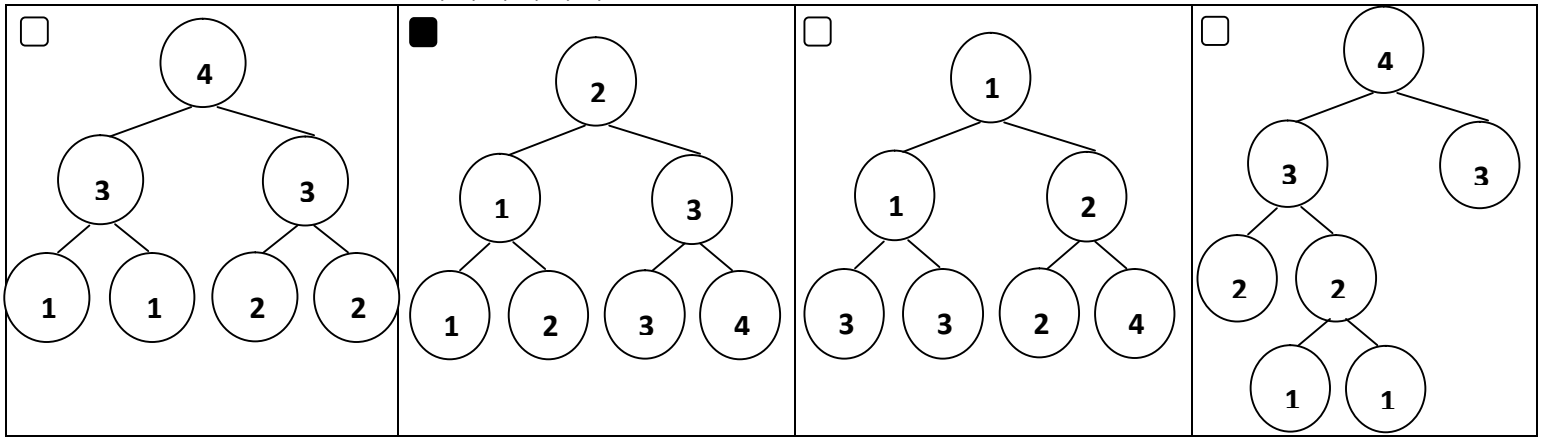
0,5 2. Etant donnée la fonction suivante:

```
int Morris(int a, int b) {
if (a==0) return 1;
return Morris(a-1, Morris(a, b));
}
```

Pour Morris (2,1), combien de fois nous faisons appel à la fonction Morris ?

☐ 0	☐ 10	☐ 100	■ ∞

1 3. Dessinez l'arbre binaire de recherche (ABR) résultant des insertions successives (aux **feuilles**) des éléments de clefs 2, 1, 3, 4, 3, 1, 2



2 4. Générer les parcours en largeur, et en profondeur (préfix, infix et postfix) de cet arbre.

Largeur :	Préfix :	Infix :	Postfix :
2, 1, 3, 1, 2, 3, 4	**2, 1, 1, 2, 3, 3, 4**	**1, 1, 2, 2, 3, 3, 4**	**1, 2, 1, 3, 4, 3, 2**

1 5. Le parcours en profondeur d'un arbre binaire correspond à un fonctionnement de :

☐ File (First In First Out)	■ Pile (First In Last Out)	☐ Liste chaînée	☐ Tableau

Dans le cas du parcours en profondeur, on va appeler récursivement la procédure avec les fils gauche et droit avant de passer à la suite. Le premier appel terminé de la procédure sera donc pour une feuille, et le dernier pour la racine : on a bien un comportement en First In (la racine) Last Out. C'est le parcours en largeur qui a un comportement de file.

1 6.

```
procedure Mystere(A : in Arbre) is
        Tmp : Arbre;
        begin
           if (A != Arbre_vide)
             if Hauteur(A.Gauche) > Hauteur(A.Droit) then
               Tmp := A.Gauche
               A.Gauche := A.Droit;
               A.Droit := Tmp;
             end if;
             Mystere(A.Gauche);
             Mystere(A.Droit);
           end if;
        end Mystere;
```

Que fait la procédure **Mystere()** ci-dessus ? On suppose qu'on dispose de la fonction **Hauteur()** qui renvoie la hauteur de l'arbre s'il est non vide.

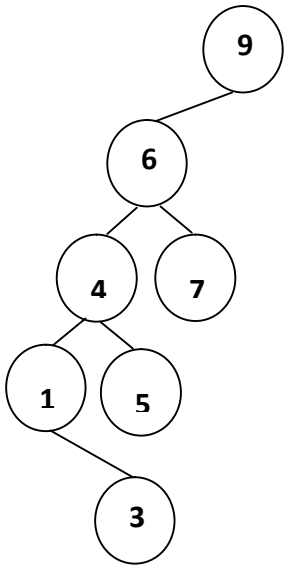
☐ Elle échange le fils gauche et le fils droit de la racine si la hauteur du fils gauche est supérieure à la hauteur du fils droit, et les autres nœuds restent à leur place.
☐ Elle échange le fils gauche et le fils droit de la racine si la hauteur du fils gauche est supérieure à la hauteur du fils droit, et du coup les autres nœuds sont aussi échangés.
☐ Elle échange le fils gauche et le fils droit de tous les nœuds dont la hauteur du fils gauche est supérieure à la hauteur du fils droit, et les autres nœuds restent à leur place.
■ Elle échange le fils gauche et le fils droit de tous les nœuds dont la hauteur du fils gauche est supérieure à la hauteur du fils droit, et du coup les autres nœuds sont aussi échangés.

1 7. Voici une liste aléatoire de 10 éléments.

5	**3**	**1**	**4**	**7**	**6**	**9**

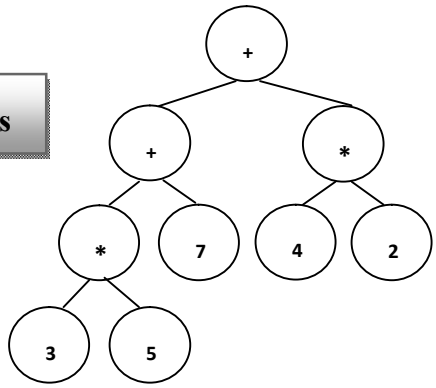
On s'intéresse aux arbres binaires de recherche.

Construire l'arbre binaire de recherche par adjonction des valeurs à la **racine**, dans l'ordre de la liste.



Exercice 2 :

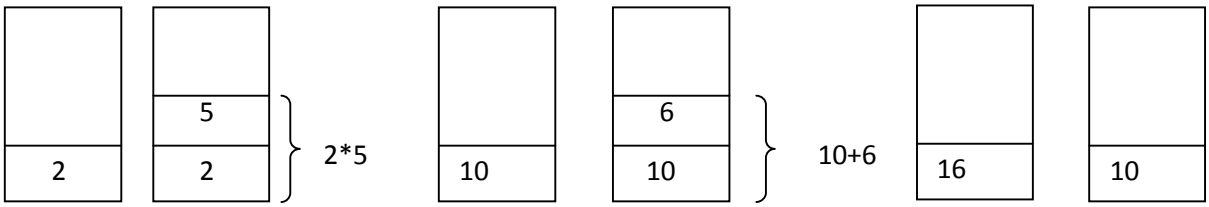
1. **Arbre binaire d'expression :** **0,5 pts**
2. **Algorithme d'évaluation d'une expression postfixe :** **3,5 pts**




```
int Evaluation(char *expr){
int i, op1, op2;
pile_t p=creerPile();
        for(i=0; expr[i]!=#; i++){
                switch(expr[i]){
                case '+' :
                        op2=depiler(p);
                        op1=depiler(p);
                        empiler(p, op1+op2);
                        break;
                case '-' :
                        op2=depiler(p);
                        op1=depiler(p);
                        empiler(p, op1-op2);
                        break;
                case '*' :
                        op2=depiler(p);
                        op1=depiler(p);
                        empiler(p, op1*op2);
                        break;
                case '/' :
                        op2=depiler(p);
                        op1=depiler(p);
                        empiler(p, op1/op2);
                        break;
                default:
                        op1=empiler(expr[i])
                }
        }
return depiler(p);
}
```

3. **La trace d'exécution pour l'expression : (2*5) +3 :** **01 pts**

(2*5)+6➔ 25*6+



Exercice 3 :

1. Les proprieties des ABR — 0,5 pts

Un arbre binaire de recherche est un arbre binaire ayant les propriétés suivantes (définition récursive):

- Tous les nœuds du sous arbre gauche de la racine ont des valeurs inférieures ou égales à la valeur de la racine.
- Tous les nœuds du sous arbre droit de la racine ont des valeurs strictement supérieures à la valeur de la racine.
- Les sous-arbre gauche et droit sont eux mêmes des arbres binaires de recherche.

2. La structure ARB — 0,5 pts

```
Type ABR= pointeur( Noeud)
Noeud= structure
Valeur= entier
Fg = ABR ; {Fils gauche}
Fd = ABR ; {Fils droit}
Fin
```

3. Création d'un ABR contenant deux nœuds X et Y — 01 pts

```
ABR  crée_arbre ( int X,Y){
{A ,B: ABR
        Si X>Y
                A.valeur = X ;
                A.Fg=B;
                B.valeur=Y ;
        Sinon
                A.valeur = Y ;
                A.Fg=B;
                B.valeur=X;
        Fsi
                A.Fd = Null;
                B.Fd=Null
                B.Fg=Null
return A ;}
```

4. Le type de parcours: — 0,5 pts

Le parcours infixe

5. Affichage des valeurs des nœuds d'un ABR de manière croissante : — 02 pts

Le principe du parcours infixe.

```
        void  Affiche_Croissant (ABR A )
        {
                Si ( A.Fg != NULL)
                        Affiche_Croissant ( A.Fg ) ;
                        afficher (A.valeur ) ;
                Si  ( A.Fd != NULL)
                        Affiche_Croissant ( A.Fd ) ;
                        afficher (A.valeur ) ;
        }
```

6. Adjonction d'une feuille

1,5 pts

```
fonction Ajoute_entier (X :entier, A : ABR) : ABR
{
        Si A = Vide Alors
                A←Créer ABR()
                A.Valeur ←X
                A.Fg ← Vide
                A.Fd← Vide
        Sinon
                Si X <A.Valeur Alors
                        Ajoute_entier (X,A.Fg)
                Sinon
                        Ajoute_entier (X,A.Fd)
                Fin Si
        Fin Si
Return (A)
}
```

7. La fonction *Adresse_noeud*

02 pts

```
Adresse_noeud ( n :entier , A : ABR ) : *ABR
{
        Si ( A.valeur = n)
                return A ;
        Si  (n <= A.valeur )
                {
                Si ( A.Fg != NULL)
                        return Adresse_noeud (n , A.Fg ) ;
                Sinon
                        return NULL;
                }
        Sinon
                 {
                Si  (A.Fd != NULL)
                        return Adresse_noeud (n , arbre->d r o i t e ) ;
                Sinon
                return NULL;
                }
}
```