8 9 1 2 7 6 4 3.

Le parcours en profondeur d'abord peut se programmer avec une boucle `while` et une liste qui contient des sous-arbres en attente d'être parcourus. On commence par mettre dans la liste un unique élément : l'arbre `a`. Puis, tant que la liste n'est pas vide, on examine son premier élément. Si cet arbre est vide, on le retire de la liste, si c'est un arbre singleton, on visite son unique nœud, sinon on le décompose en trois arbres plus petits : son sous-arbre gauche, son sous-arbre droit et l'arbre singleton constitué de sa racine et on ajoute ces trois arbres à la liste. Comme ces trois arbres doivent être parcourus avant tous les autres arbres de la liste, cette liste doit être une pile.

Quand on fait un parcours infixe, on parcourt ces trois arbres dans l'ordre : gauche, racine, droit, il suffit de les empiler dans l'ordre inverse.

```
Pile p = Pile.empty ();
p.push(a);
 while (!(p.testempty())) {
  Arbre b = p.top ();
  p.pop();
  if (b != null) {
   if (singleton(b))
    System.out.print(b.val + " ");
    else {
     p.push(b.droit);
     p.push(new Arbre(b.val,null,null));
     p.push(b.gauche);}}}
```

En changeant l'ordre dans lequel on empile les trois arbres, on peut programmer le parcours préfixe ou postfixe.

On peut éviter d'utiliser une pile en programmant le parcours de manière récursive

```
static void parcours (Arbre a) {
 if (a != null) {
  parcours(a.gauche);
  System.out.print(a.val + " ");
  parcours(a.droit);}}
```

Exercice 9.2

Écrire une fonction qui effectue un parcours préfixe d'un arbre et une fonction qui en effectue un parcours postfixe.