

Inordre

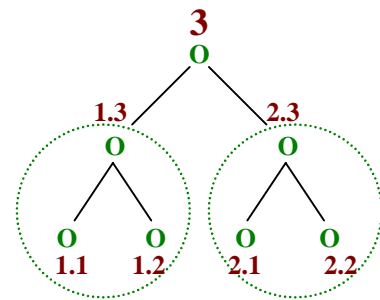
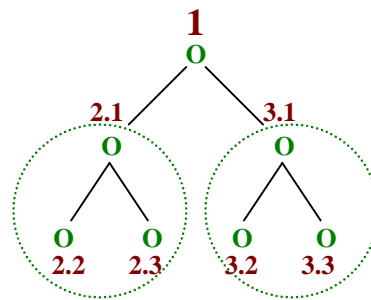
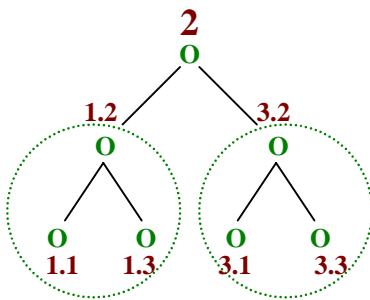
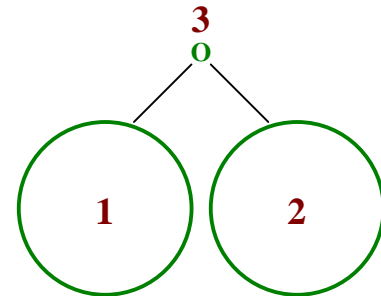
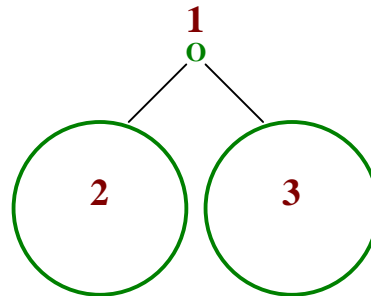
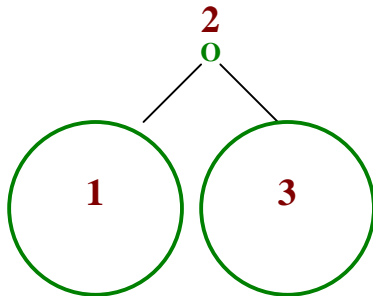
Sous-arbre gauche
Lui-même
Sous-arbre droit

Préordre

Lui-même
Sous-arbre gauche
Sous-arbre droit

Postordre

Sous-arbre gauche
Sous-arbre droit
Lui-même



```
void ParcoursInordre( TypeNoeud *ptr )
{
    if( !ptr ) return;
    ParcoursInordre( ptr->gauche );
    cout << ptr->data << endl;
    ParcoursInordre( ptr->droit );
}
```

```
void ParcoursPreordre ( TypeNoeud *ptr )
{
    if( !ptr ) return;
    cout << ptr->data << endl;
    ParcoursPreordre( ptr->gauche );
    ParcoursPreordre( ptr->droit );
}
```

```
void ParcoursPostordre( TypeNoeud *ptr )
{
    if( !ptr ) return;
    ParcoursPostordre( ptr->gauche );
    ParcoursPostordre( ptr->droit );
    cout << ptr->data << endl;
}
```

OU

```
void ParcoursInordre( TypeNoeud *ptr )
{
    if( ptr->gauche )
        ParcoursInordre( ptr->gauche );
    cout << ptr->data << endl;
    if( ptr->droit )
        ParcoursInordre( ptr->droit );
}
```

```
void ParcoursPreordre ( TypeNoeud *ptr )
{
    cout << ptr->data << endl;
    if( ptr->gauche )
        ParcoursPreordre( ptr->gauche );
    if( ptr->droit )
        ParcoursPreordre( ptr->droit );
}
```

```
void ParcoursPostordre( TypeNoeud *ptr )
{
    if( ptr->gauche )
        ParcoursPostordre( ptr->gauche );
    if( ptr->droit )
        ParcoursPostordre( ptr->droit );
    cout << ptr->data << endl;
}
```

```
typedef struct nd
{
    int data;
    struct nd *gauche;
    struct nd *droit;
} TypeNoeud;
```

```
class TypeNoeud
{
public :
    int data;
    TypeNoeud *gauche;
    TypeNoeud *droit;
};
```

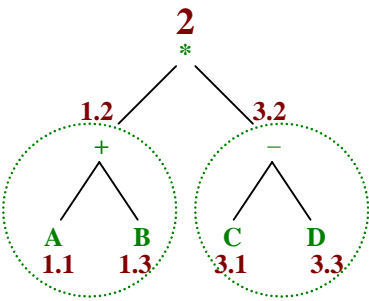
Notation infixée

$A + B - C$
 $A * B / C$
 $C * (A + B) / D$
 $(A + B) * C$
 $A + B * C$
 $(A + B) * (C - D)$
 $(A + B) * C - D$
 $A + B * (C - D) / (E + F)$

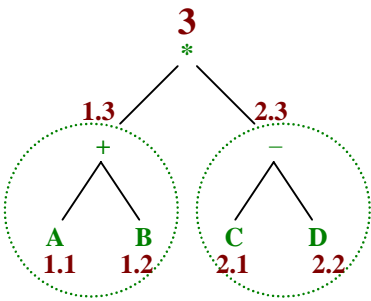


Notation postfixée

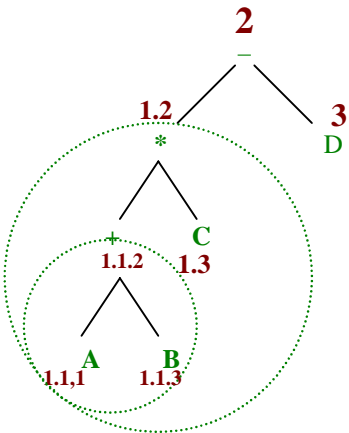
$AB+C-$
 $AB*C/$
 $CAB+D/*$
 $AB+C*$
 $ABC*+$
 $AB+CD-*$
 $AB+C*D-$
 $ABCDEF+/-*+$



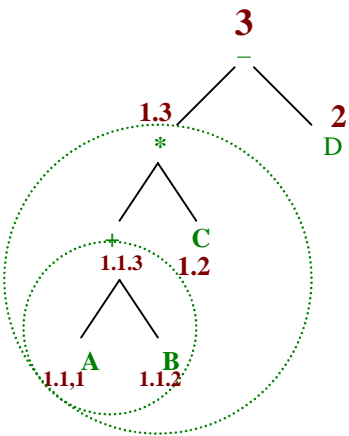
$(A + B) * (C - D)$



$AB + CD - *$



$(A + B) * C - D$



$AB + C * D -$

Méthode d'arrêt de la récursive (cas de base)

```
void ParcoursPostordre( TypeNoeud *ptr )
{
    if( !ptr ) return;
    ParcoursPostordre( ptr->gauche );
    ParcoursPostordre( ptr->droit );
    cout << ptr->data << endl;
}
```

versus

```
void ParcoursPostordre( TypeNoeud *ptr )
{
    if( ptr->gauche )
        ParcoursPostordre( ptr->gauche );
    if( ptr->droit )
        ParcoursPostordre( ptr->droit );
    cout << ptr->data << endl;
}
```

La première méthode économise la redondance de la référence du pointeur (celle du test if) mais elle coûte N+1 appels inutiles pour les enfants des feuilles qui ne sont que des pointeurs null. La fonction est appelée et retourne aussitôt grâce à (**if(!ptr) return;**) mais cela a coûté l'appel de la fonction, soit le stockage de la position dans le programme et de la valeur des variables locales. En contrepartie, cette implémentation présente l'avantage d'une meilleure sécurité dans le cas où la tête passée en paramètre à l'appel initial serait déjà nulle. Elle est également plus lisible pour l'humain et génère un code plus court.

La seconde version ne traite pas le cas particulier de la tête nulle et mais présente l'avantage de faire 2 fois moins d'appels récursifs.

Les deux versions auront un segfault si les données présentées ne terminent pas les hiérarchies par des pointeurs null (toujours les pointeurs à zéro lorsqu'ils ne pointent pas vers un enfant).

Notation infixée versus notation postfixée

Les caractéristiques de la notation infixée sont la facilité de lecture pour l'être humain et la précedence des opérateurs prédéfinie. Un parcours inordre d'un arbre arithmétique permet d'obtenir la notation infixée et d'effectuer les opérations dans le bon ordre pour l'obtention du résultat (en substituant les variables par des valeurs). Les parenthèses correspondent aux cercles des sous-arbres sur le schéma ci-haut, et les parenthèses cascadiées sont des cercles concentriques. Évidemment, de telles parenthèses ne sont requise que si l'opérateur à la tête du sous-arbre en question est un + ou un -, dans le cas d'un sous-arbre dont la tête est un * ou un /, leur forte précedence rend l'écriture des parenthèses facultative.

Les caractéristiques de la notation postfixée sont l'absence des parenthèses qui ne sont plus nécessaire et la définition de la précedence totalement contrôlée par la position des éléments dans la chaîne. Cette notation est considérée comme plus adaptée aux piles utilisées par les ordinateurs (revoir la section sur les listes).

```
void ParcoursInfixe( TypeNoeud *ptr )
{
    if( !ptr ) return;
    // ou seulement if( ptr->signe ) pour avoir toutes les parenthèses.
    if( ptr->signe == PLUS || ptr->signe == MOINS )
        cout << " ( ";

    ParcoursInfixe( ptr->gauche );

    switch( ptr->signe )
    {
        case DATA :      cout << ptr->data << " "; break;
        case PLUS :       cout << " + ";          break;
        case MOINS :      cout << " - ";          break;
        case MULT :       cout << " * ";          break;
        case DIV :        cout << " / ";          break;
        default :         return;
    }

    ParcoursInfixe( ptr->droit );

    if( ptr->signe == PLUS || ptr->signe == MOINS )
        cout << " ) ";
}
```

```
#define DATA 0
#define PLUS 1
#define MOINS 2
#define MULT 3
#define DIV 4

typedef struct nd
{
    int data;
    char signe;
    struct nd *gauche;
    struct nd *droit;
} TypeNoeud;
```

Exercices

Reconstituer l'arbre de l'expression infixée $C*(A+B)/D$

Reconstituer l'arbre de l'expression postfixée $ABCDEF+/-*+$

Reconstituer l'arbre dont le résultat des parcours sont les suivants :

Parcours inordre = $7\ 1\ 11\ 5\ 3\ 4\ 9\ 2\ 27$

Parcours postordre = $4\ 3\ 1\ 7\ 5\ 11\ 27\ 9\ 2$