TD1.6 – Preuves de correction et de terminaison

Objectifs

À la fin de cette séance, vous devriez être capable de :
- proposer et représenter un invariant de boucle ;
- spécifier un algorithme récursif ;
- prouver la correction d'un algorithme récursif simple ;
- prouver la terminaison d'un algorithme récursif.

Exercice 1 : Dessins d'invariants

Rappel : un invariant est une propriété qui est maintenue par chaque itération : si elle est vraie au début d'une itération quelconque, alors elle est encore vraie à la fin de cette itération.

On s'intéresse à l'algorithme suivant :

```
Maximum(T[0..n−1]) :
    /* Postcondition : max est une valeur ≥ à toutes celles dans T */
    max := T[0]
    pour i := 1 à n − 1 :
        si T[i] > max :
            max := T[i]
```

1. Proposez un ou plusieurs invariants et représentez-les graphiquement.
2. On rappelle qu'un invariant n'est utile qu'à condition :
   — qu'il soit vrai au moment où on entre dans la boucle
   — qu'il permette de déduire la postcondition en sortie de boucle
   Parmi les invariants que vous avez proposés, lesquels ne vérifient pas tous ces critères ?
3. Si besoin, proposez un nouvel invariant qui vérifie tous les critères cités ci-dessus.
4. Une fois que vous avez obtenu un invariant utile, démontrez qu'il s'agit bien d'un invariant.
5. Mêmes questions pour l'algorithme :

```
Minimum(T[0..n−1]) :
    /* Postcondition : T[i] est le minimum de T */
    i := 0
    j := n − 1
    tant que i < j :
        si T[i] < T[j] :
            j := j − 1
        sinon :
            i := i + 1
```

6. ***Bonus, à ne faire qu'après l'exercice 2 :*** mêmes questions pour ce troisième algorithme.

```
Pangramme(T[0..n−1], k) :
    /* Postcondition : renvoie vrai si le tableau T contient au moins un
       exemplaire de chaque nombre entre 0 et k */
    Soit V un tableau de booléens indexé de 0 à k.
    pour j := 0 à k :
        V[j] := faux
    pour i := 0 à n − 1 :
        V[T[i]] := vrai
    pour j := 0 à k :
        si (non V[j]) :
            renvoyer faux
    renvoyer vrai
```

Exercice 2 : Preuve d'algorithme récursif

On a écrit l'algorithme suivant pour déterminer si un arbre binaire est *parfait*, c'est-à-dire que toutes ses feuilles sont à la même profondeur, et tous ses autres nœuds ont deux fils non vides.

```
Parfait(A) :
    si A est une feuille :
        renvoyer 0
    sinon si FilsGauche(A) est vide ou FilsDroit(A) est vide :
        renvoyer -1
    sinon :
        hg := Parfait(FilsGauche(A))
        hd := Parfait(FilsDroit(A))
        si hg ≥ 0 et hd ≥ 0 et hg = hd :
            renvoyer hg + 1
        sinon :
            renvoyer -1
```

On cherche à démontrer que cet algorithme récursif est correct.

1. Dessinez à quoi ressemble un arbre parfait.
2. Écrivez la spécification (précondition, postcondition) de la fonction récursive Parfait.
3. Vérifiez que les appels récursifs vérifient votre précondition.
4. Démontrez que les cas de base vérifient votre postcondition.
5. Démontrez que les cas récursifs vérifient votre postcondition.
   (Rappel : vous pouvez pour cela supposer que les appels récursifs sont corrects.)
6. Démontrez la terminaison de cette fonction.

Membres du groupe et responsabilités

	Appliquer des méthodes algorithmiques connues (algorithme récursif, parcours d'arbre)	Gérer le temps
	Évaluer l'efficacité des algorithmes	Réguler les prises de parole
	Rechercher des cas problématiques Déterminer des invariants	S'assurer que tous comprennent et sont en accord avec le rendu
	Représenter schématiquement les invariants des algorithmes	Rédiger le rendu

Exercice 1 : Dessins d'invariants

```
Minimum(T[0..n-1]) :
    /* Postcondition : T[i] est le minimum de T */
    i := 0
    j := n - 1
    tant que i < j :
        si T[i] < T[j] :
            j := j - 1
        sinon :
            i := i + 1
```

1. Proposez un ou plusieurs invariants pour l'algorithme ci-dessus et représentez-les graphiquement.
2. Parmi les invariants proposés, lesquels sont utiles pour démontrer que cet algorithme est correct ? Lesquels sont inutiles et pourquoi ?
3. Pour l'un de vos invariants utiles, démontrez qu'il s'agit bien d'un invariant.