Algorithmes et Structures de Données
DS1 – Vendredi 15 novembre 2024
Correction

1. Sudoku (8 pts)

```
Const n = 3
Type tab-bool = tableau[1..n*n] de booléens
     Sudoku = tableau[1..n*n,1..n*n] d'entiers
```

1.1. Ecrire en pseudo-langage la fonction `vérifie-tab(t : tab-bool) : booléen`, qui vérifie que chaque booléen de `t` est égal à `vrai`. Elle renvoie `vrai` dans ce cas, `faux` sinon.

```
Fonction vérifie-tab(t : tab-bool) : booléen
Var i:entier
Début
i←1
TantQue t[i]=vrai et i<=n*n Faire
    i←i+1
FinTantQue
Retourne(i=n*n+1)
Fin
```

1.2. Ecrire en pseudo-langage la fonction `vérifie-ligne(s : sudoku ; l : entier) : booléen`, qui vérifie que la ligne `l` est valide.

```
Procédure intialise(S t : tab-bool)
Var i:entier
Début
Pour i←1 à n*n inc +1 Faire
    t[i]←faux
FinPour
Fin

Fonction vérifie-ligne(s : sudoku ; l : entier) : booléen
Var i:entier
    t :tab-bool
Début
initialise(t)
Pour i←1 à n*n inc +1 faire
    Si s[l,i]<>0
       Alors t[s[l,i]]←vrai
    FinSi
FinPour
Retourne(vérifie-tab(t))
Fin
```

1.3. Ecrire en pseudo-langage la fonction `vérifie-colonne(s : sudoku ; c : entier) : booléen`, qui vérifie que la colonne `c` est valide.

```
Fonction vérifie-colonne(s : sudoku ; c : entier) : booléen
Var i:entier
    t :tab-bool
Début
initialise(t)
Pour i←1 à n*n inc +1 faire
    Si s[i,c]<>0
       Alors t[s[i,c]]←vrai
    FinSi
FinPour
Retourne(vérifie-tab(t))
Fin
```

1.4. Ecrire en pseudo-langage la fonction `vérifie-bloc(s : sudoku ; i, j : entier) : booléen`, qui vérifie que le bloc (dont le coin supérieur gauche est en `i,j)` est valide.

```
Fonction vérifie-bloc(s:sudoku;i,j:entier):booléen
Var l,c :entier
    t :tab-bool
Début
initialise(t)
Pour l←i à i+n-1 inc +1 faire
    Pour c← j à j+n-1 inc +1 faire
       Si s[l,c]<>0
           Alors t[s[l,c]]←vrai
       FinSi
    FinPour
FinPour
Retourne(vérifie-tab(t))
Fin
```

3.5. Ecrire en pseudo-langage la fonction `vérifie-sudoku(s : sudoku) : booléen`, qui vérifie que la grille complète du sudoku est valide.

```
Fonction vérifie-sudoku(s:sudoku):booléen
Var i,j,erreur:entier
    t :tab-bool
Début
erreur←0
Pour i←1 à n*n inc +1 faire
    Si Vérifie-ligne(s,i)=faux
       Alors erreur←erreur+1
    Finsi
FinPour
Pour i←1 à n*n inc +1 faire
    Si Vérifie-colonne(s,i)=faux
       Alors erreur←erreur+1
    Finsi
FinPour
Pour i←1 à n*n inc +n faire
    Pour j←1 à n*n inc +n faire
        Si Vérifie-bloc(s,i,j)=faux
           Alors erreur←erreur+1
        Finsi
    FinPour
FinPour
Retourne(erreur=0)
Fin
```

3.6. On souhaite charger une grille de sodoku à résoudre à partir du fichier texte de nom `nomfich`.
Chaque ligne du fichier est une ligne du sudoku dont les chiffres sont séparés par un `' '`.
Les cases vides contiennent `'0'`. On considère que le fichier est bien formé.
Ecrire en pseudo-langage la procédure `charge-sudoku(E nomfich : chaine, S t : sudoku)`.

```
Procedure charge-sudoku(E nomfich : chaine, S s : sudoku)
Var l,c,i : entier
    f : FT
    ligne : chaine
Debut
    f←OuvrirEnLecture(nomfich)
    Pour l←1 à n*n inc +1 Faire
       ligne←lireChaine(f)
       i←1
       Pour C←1 à n*n inc +1 Faire
           s[l,c]← chaine2entier(mot-suiv(ligne,i))
       FinPour
    FinPour
    Fermer(f)
Fin
```

3.7. On souhaite sauvegarder la grille de sodoku résolue dans un fichier texte dont on demandera le nom à l'utilisateur. Chaque ligne du fichier est une ligne du sudoku dont les chiffres sont séparés par un `' '`.
Ecrire en pseudo-langage la procédure `sauvegarde-sudoku(E s : sudoku)`.

```
Procedure sauvegarde-sudoku(E s : sudoku)
Var l,c : entier
    f : FT
    nomfich, ligne : chaine
Debut
    écrire ('Entrez le nom du fichier de sauvegarde')
    lire(nomfich)
    f← créerFichier(nomfich)
    Pour l←1 à n*n inc +1 Faire
       ligne←''
       Pour c←1 à n*n inc +1 Faire
              concatener(ligne,entier2Chaine(s[l,c]),' ')
       FinPour
       f←écrireChaine(ligne)     {ou écrireChaine(f,ligne)}
    FinPour
    Fermer(f)
Fin
```

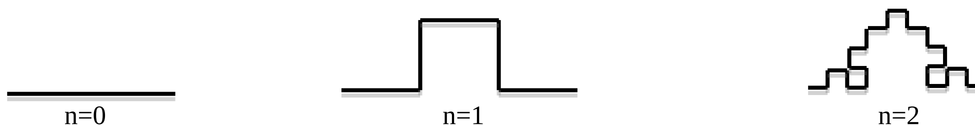
2. Fonctions récursives et pile (6 pts)

```
Fonction z(n : entier) : entier
Var res : entier
Début
Si n>0
       alors res←n-1
       sinon res←z(z(n+2){@1}){@2}
FinSi
Retourner(res)
Fin
```

Simuler la pile sur l'appel `écrire(z(-5)){@0}` dans le programme principal.

```
@2     n=1          res=0
@1     n=2          res=1
@2     n=0          res=0
@2     n=1          res=0
@1     n=2          res=1
@2     n=0          res=0
@2     n=1          res=0
@1     n=2          res=1
@2     n=0          res=0
@1     n=1          res=0
@1     n=-1         res=0
@1     n=-3         res=0
@0     n=-5         res=0
```

1. Fractale : Carré de von Koch (6 pts)



Ecrire en pseudo-langage la **procédure récursive** `dessine-coté-VonKoch(E lg,n:entier)` qui permet de dessiner un côté du carré de longueur `lg` et de niveau `n`.
Comme outil de dessin, nous avons

- une procédure `trace-ligne(E lg:entier)` qui trace une ligne de longueur `lg` à partir du point courant (où se trouve le stylo) et dans la direction courante. `Trace-ligne` déplace le stylo.
- Une procédure `tourne(E d:entier)` qui tourne le stylo d'un angle donné par `d` en degré vers la gauche si d >0 (vers la droite sinon).

```
Procédure dessine-coté-VonKoch(E lg, n : entier)
Début
Si n=0
   Alors trace-ligne(lg)
   Sinon lg ← lg div 3
         dessine-coté-VonKoch(lg,n-1)
         tourne(90)
         dessine-coté-VonKoch(lg,n-1)
         tourne(-90)
         dessine-coté-VonKoch(lg,n-1)
         tourne(-90)
         dessine-coté-VonKoch(lg,n-1)
         tourne(90)
         dessine-coté-VonKoch(lg,n-1)
FinSi
Fin
```