```
A <- 42
B <- 21
C <- A + B
afficher(C)
```

```
63
```

```
laurent@client-linux:~/Bureau/demo$ cd ..
laurent@client-linux:~/Bureau$ mkdir permutation
laurent@client-linux:~/Bureau$ cd permutation
laurent@client-linux:~/Bureau/permutation$ echo 57 > A
laurent@client-linux:~/Bureau/permutation$ echo 17 > B
laurent@client-linux:~/Bureau/permutation$ cat A > B
laurent@client-linux:~/Bureau/permutation$ cat B
57
laurent@client-linux:~/Bureau/permutation$ echo 17 > B
laurent@client-linux:~/Bureau/permutation$ # permuter les valeurs de A
et B (B devrait valoir 57 et A, 17) (sans la commande echo)
laurent@client-linux:~/Bureau/permutation$
```

```
17
laurent@client-linux:~/Bureau/permutation$ cat A B C
57
17
57
laurent@client-linux:~/Bureau/permutation$ cat A B C > Fichier
laurent@client-linux:~/Bureau/permutation$ cat Fichier
57
17
57
laurent@client-linux:~/Bureau/permutation$ cat A > C
laurent@client-linux:~/Bureau/permutation$ cat B > A
laurent@client-linux:~/Bureau/permutation$ cat C > B
laurent@client-linux:~/Bureau/permutation$
```

```
laurent@client-linux:~/Bureau/apprendre$ ls /usr/bin
'['                                     padsp
 7z                                     pager
 7za                                    pa-info
 7zr                                    pamon
 pacat                                      zstdmt
 pacmd                                      zutty
 pactl
laurent@client-linux:~/Bureau/apprendre$ echo a b c d | wc
      1       4       8
laurent@client-linux:~/Bureau/apprendre$ echo a b c d
a b c d
laurent@client-linux:~/Bureau/apprendre$ echo a b c d| wc
      1       4       8
laurent@client-linux:~/Bureau/apprendre$ echo a b c d| wc -w
4
laurent@client-linux:~/Bureau/apprendre$ ls /usr/bin | wc -w
1117
laurent@client-linux:~/Bureau/apprendre$
```

AlgoBox 1.1.1 [modifié] : sanstitre

Nouveau | Ouvrir | Sauver | Tester | Mode normal | Mode éditeur texte

À propos de cet algorithme:

Paramètres de cet algorithme: Arrondi auto : vrai | Changer | Extension utilisée:

```
FONCTIONS_UTILISEES
VARIABLES
    A EST_DU_TYPE NOMBRE
    B EST_DU_TYPE NOMBRE
DEBUT_ALGORITHME
    LIRE A
    SI (A>0) ALORS
        DEBUT_SI
        AFFICHER "1"
        FIN_SI
        SINON
            DEBUT_SINON
            SI (A==0) ALORS
                DEBUT_SI
                AFFICHER "0"
                FIN_SI
                SINON
                    DEBUT_SINON
                    AFFICHER "-1"
                    FIN_SINON
            FIN_SINON
FIN_ALGORITHME
```

Modifier Ligne

Supprimer Ligne/Bloc

- Pour utiliser une variable,il faut d'abord la déclarer (bouton "Déclarer nouvelle variable").
- Pour ajouter un nouvel élément à l'algorithme,il faut d'abord insérer une nouvelle ligne (bouton "Nouvelle Ligne"),puis cliquer sur un des boutons disponibles dans les panneaux disponibles en bas de la fenêtre.

Tester Algorithme

Nouvelle Ligne

Opérations standards | Utiliser une fonction numérique | Dessiner dans un repère | Fonctions locales

+ Déclarer nouvelle variable
+ Ajouter LIRE variable
+ AFFECTER valeur à variable
+ Ajouter AFFICHER Variable
+ Ajouter AFFICHER Message
+ Ajouter AFFICHER Calcul
+ Ajouter SI...ALORS
+ Ajouter POUR...DE...A
+ Ajouter TANT QUE...
Pause
Commentaire

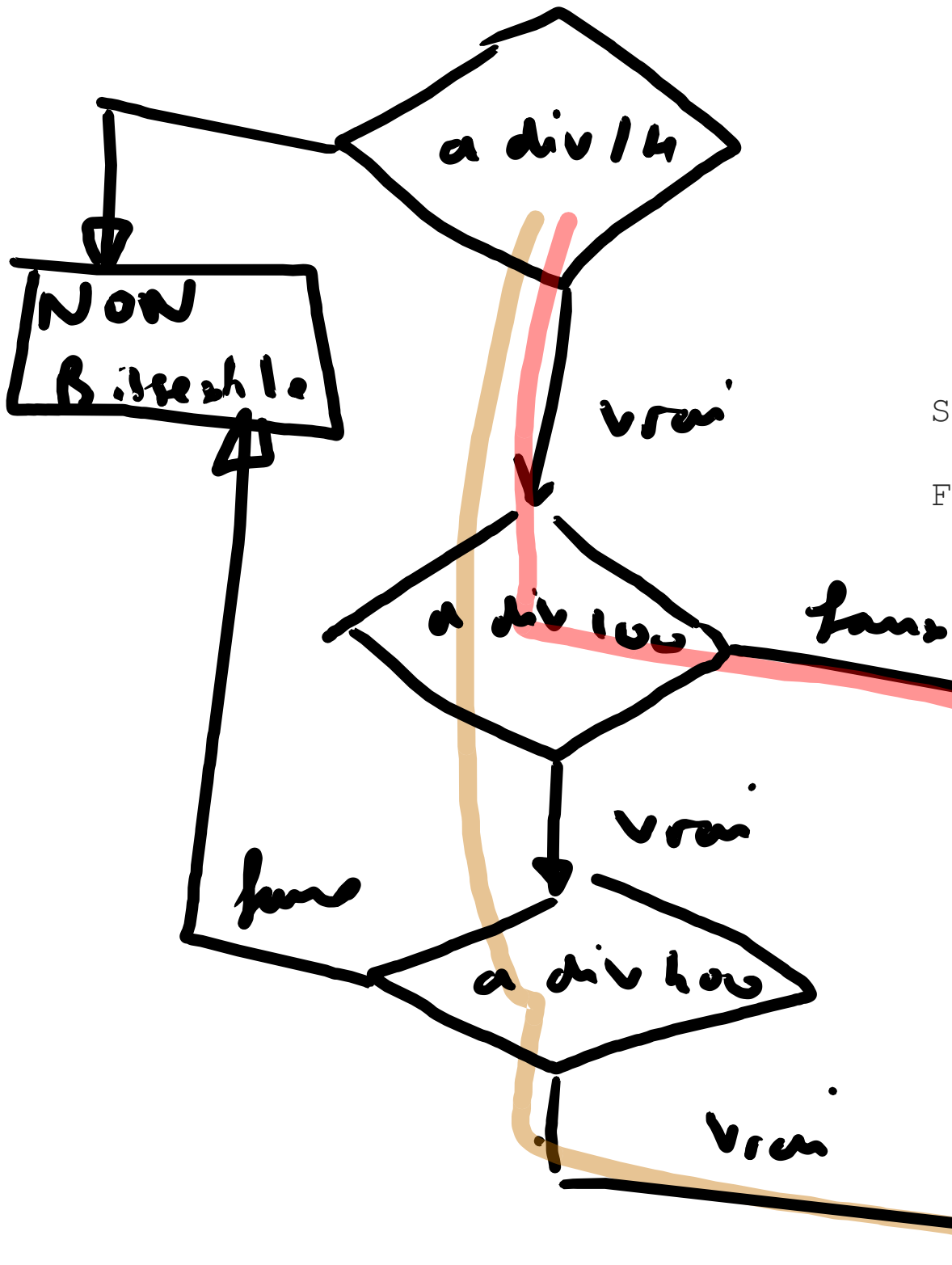
Bissextile

Année divisible par 4

non divisible par 100

divisible par 400

a : div/4 ET (non div/100 OU div/400)

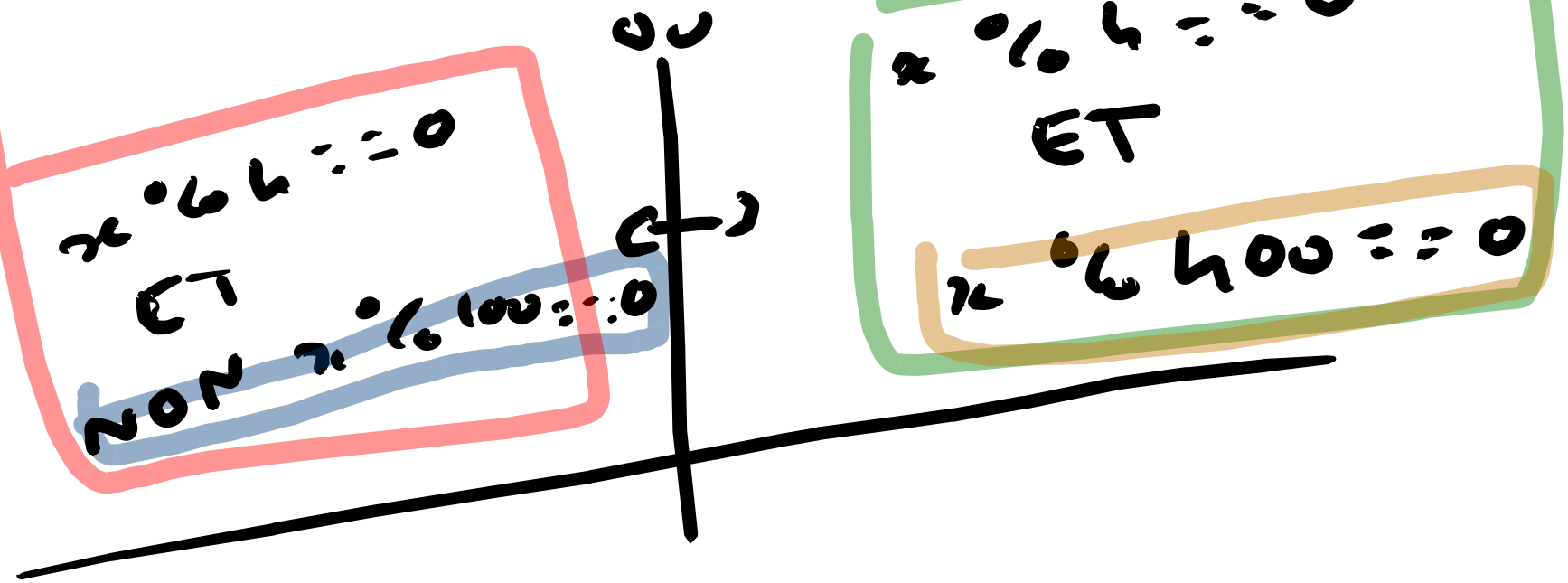



```
SI a % 4 == 0 ALORS
      SI a % 100 == 0 ALORS
            SI a % 400 == 0 ALORS
                  AFFICHER "bissextile"
            SINON
                  AFFICHER "NON bissextile"
            FINSI
      SINON
            // div par 4 mais pas par 100
            AFFICHER "bissextile"
      FINSI
SINON
      AFFICHER "NON bissextile"
FINSI
```

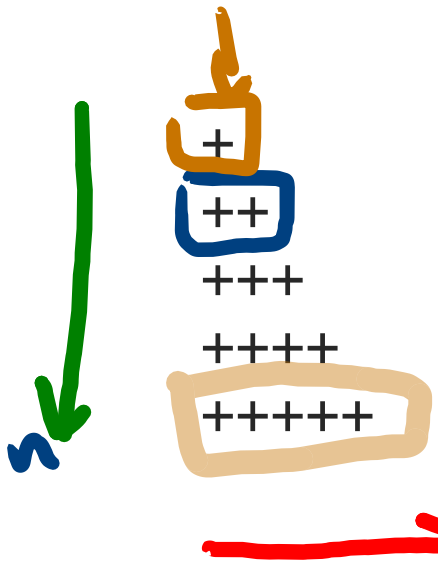
```
SI a % 4 == 0 ALORS
      SI a % 100 == 0 ALORS
            SI a % 400 == 0 ALORS
                  AFFICHER "bissextile"
            SINON
                  AFFICHER "NON bissextile"
            FINSI
      SINON
            // div par 4 mais pas par 100
            AFFICHER "bissextile"
      FINSI
SINON
      AFFICHER "NON bissextile"
FINSI
```

$$x \times (a + b) = x \times a + x \times b$$

x EST a OU x EST b



```
VARIABLES
     mois: ENTIER
     annee: ENTIER
DEBUT
     SELON mois
          cas 1 OU 3 OU 5 OU 7 OU 8 OU 10 OU 12:
               AFFICHER 31
          cas 4 OU 6 OU 9 OU 11:
               AFFICHER 30
          cas 2:
               SI annee % 4 == 0 ET (annee %400 == 0 OU annee % 100 == 0)
               ALORS
                    AFFICHER 29
               SINON
                    AFFICHER 28
               FINSI
          defaut:
               AFFICHER "ERREUR mois invalide"
     FIN_SELON
FIN
```



λ λc 1 +

λ dc l +

λ dc n +

1 + + +


0
1
2
n
? nb espace

```
VARIABLE
     i : ENTIER
DEBUT
     i <-- 1
     TANTQUE i <= 10
          AFFICHER i
          i <-- i + 1
     FIN TANTQUE
FIN
```

```
VARIABLE
     i : ENTIER
DEBUT
     i <-- 0
     TANTQUE i <= 9
          AFFICHER 2^i
          i <-- i + 1
     FIN TANTQUE
FIN
```

```
VARIABLE
     i : ENTIER
DEBUT
     POUR i ALLANT_DE 1 à 10
          AFFICHER i
     FIN POUR
FIN
```

```
VARIABLE
     i : ENTIER
DEBUT
     POUR i ALLANT_DE 0 à 9
          AFFICHER 2^i
     FIN POUR
FIN
```

Algorithme 3

```
1   FONCTIONS_UTILISEES
2   VARIABLES
3     A EST_DU_TYPE NOMBRE
4     B EST_DU_TYPE NOMBRE
5     C EST_DU_TYPE NOMBRE
6   DEBUT_ALGORITHME
7     A PREND_LA_VALEUR 3
8     B PREND_LA_VALEUR 7
9     A PREND_LA_VALEUR A + B
10    C PREND_LA_VALEUR B - A
11    A PREND_LA_VALEUR B + C
12    AFFICHER A
13    AFFICHER B
14    AFFICHER C
15  FIN_ALGORITHME
```

Algorithme 6

```
1 :   VARIABLES
2 :   x EST_DU_TYPE NOMBRE
3 :   racine EST_DU_TYPE NOMBRE
4 :   DEBUT
5 :   LIRE x
6 :   SI ( ..... ) ALORS
7 :   DEBUT_SI
8 :       racine PREND_LA_VALEUR sqrt(x)
9 :       AFFICHER racine
10:   FIN_SI
11:   FIN
```

x >= 0

Algorithme 7

```
1 : VARIABLES
2 : x EST_DU_TYPE NOMBRE
3 : y EST_DU_TYPE NOMBRE
4 : DEBUT
5 :     LIRE x
6 :     LIRE y
7 :     SI (x > y) ALORS
8 :     DEBUT_SI
9 :         AFFICHER ......
10:     SINON
11:         AFFICHER ......
12:     FIN_SI
13: FIN
```

9: x

11: y

Algorithme 8

```
VARIABLES
A EST_DU_TYPE NOMBRE
B EST_DU_TYPE NOMBRE
DEBUT
1    A PREND_LA_VALEUR 1
2    B PREND_LA_VALEUR 3
3    SI (A > 0) ALORS
    DEBUT_SI
4        A PREND_LA_VALEUR A + 1
5        B PREND_LA_VALEUR A + B
    FIN_SI
6    SI (B > 4) ALORS
    DEBUT_SI
7        B PREND_LA_VALEUR B - 1
8        A PREND_LA_VALEUR A - B
    FIN_SI
FIN
```

A:

lgn	A	B	Cond
1	1	ND	
2	1	3	
3	1	3	VRAI
4	2	3	
5	2	5	
6	2	5	VRAI
7	2	4	
8	-2	4	

Algorithme 15

```
VARIABLES
A EST_DU_TYPE NOMBRE
B EST_DU_TYPE NOMBRE
DEBUT
    A PREND_LA_VALEUR 9
    B PREND_LA_VALEUR 14
    SI ( ..... ) ALORS
    DEBUT_SI
        A PREND_LA_VALEUR A + B
    SINON
        A PREND_LA_VALEUR A - B
    FIN_SI
    AFFICHER A
FIN
```

B <= A

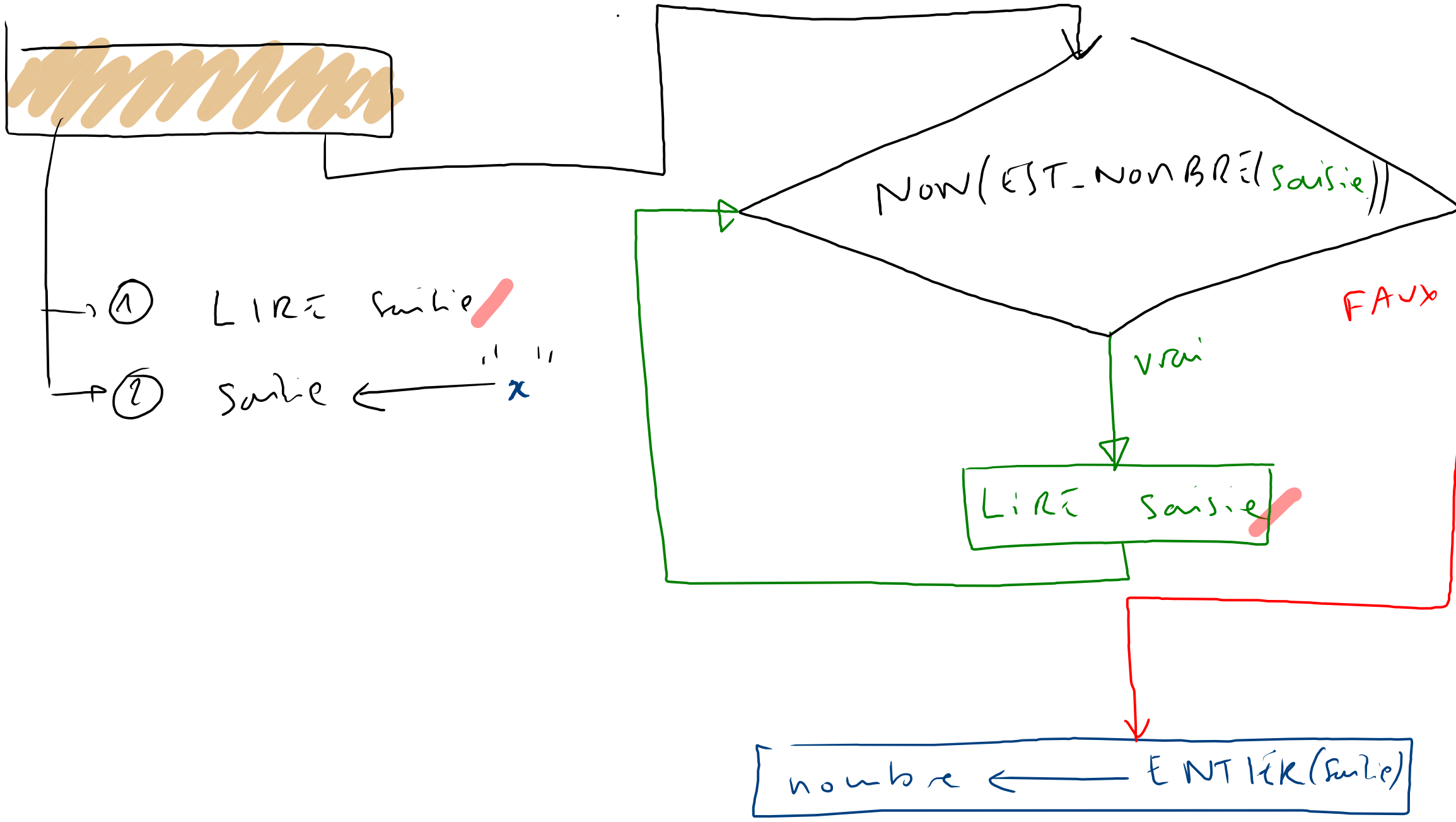
A == B

* A < B

A > B

Algorithme 16

```
VARIABLES
A EST_DU_TYPE NOMBRE
B EST_DU_TYPE NOMBRE
DEBUT
    A PREND_LA_VALEUR 4
    B PREND_LA_VALEUR 5
    SI ( ( A - B ) < 0 ) ALORS
    DEBUT_SI
        AFFICHER A
    SINON
        AFFICHER B
    FIN_SI
FIN
```


NON(EST_NOMBRE(saisie))
FAUX
VRAI
LIRE saisie
1 LIRE saisie
2 saisie ← "x"
nombre ← ENTIER(saisie)

Créer la fonction **est_pair(nombre)** qui renvoie VRAI si le nombre entier passé en paramètre est pair, FAUX sinon.

Écrire le programme principal permettant d'afficher si 2, 5, 8 et 13 sont pairs ou pas.

Exemple d'affichage

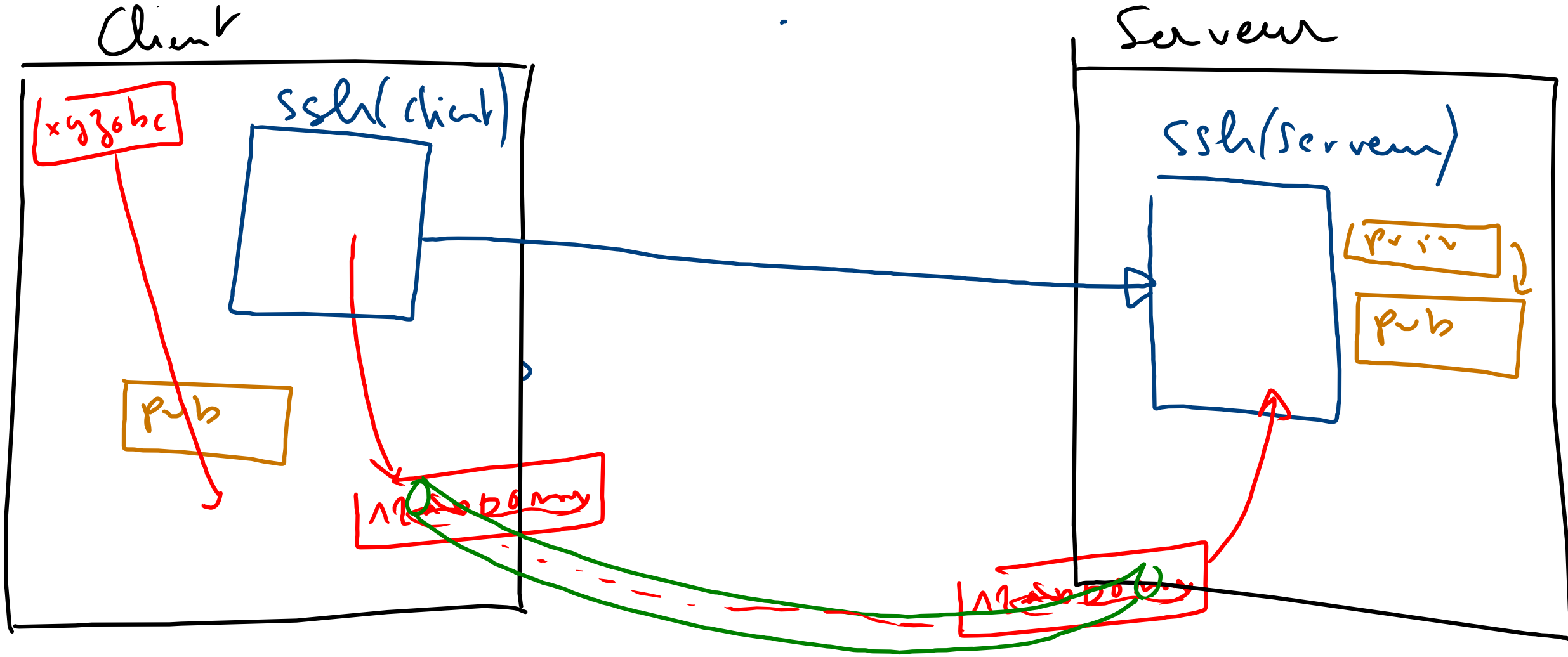
```
2 est pair
5 n'est pas pair
```

```
FONCTION est_pair(nombre : ENTIER) : renvoie type BOOLEEN
DEBUT
      RENVOYER nombre MODULO 2 == 0
FIN_FONCTION
```

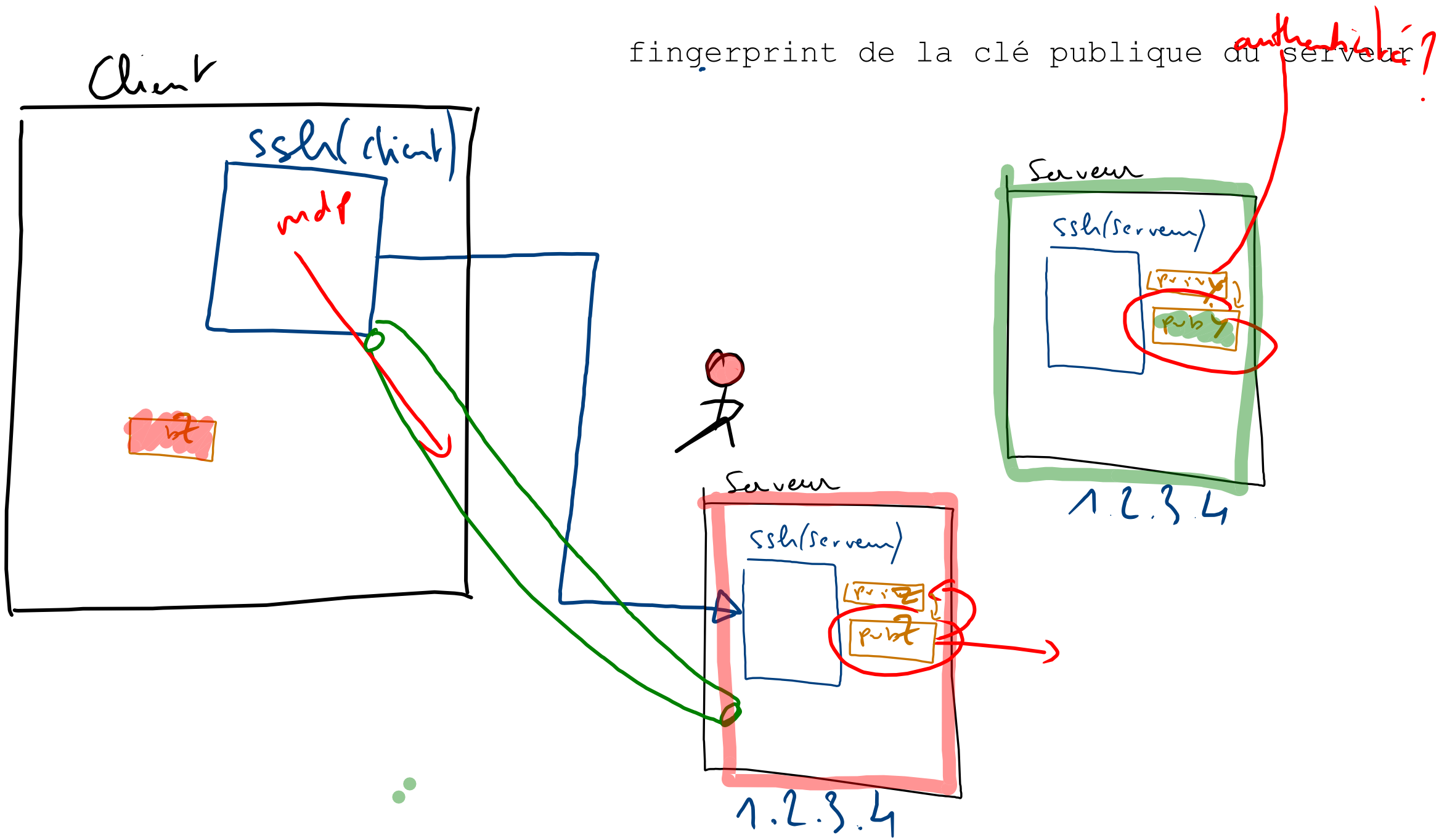
```
PROGRAMME PRINCIPAL
VARIABLES
       x : ENTIER
DEBUT
       AFFICHER("Saisir un nombre: ")
       LIRE x
       SI est_pair(x) ALORS
              AFFICHER(x & " est pair")                // x + " est pair"
       SINON
              AFFICHER(x & " est impair")
       FIN_SI

FIN
```

```
def est_pair(nombre):
        return nombre % 2 == 0

// javascript
function est_pair(nombre) {
        return nombre % 2 == 0 ;
}


//C
int est_pair(int nombre) {
        return nombre % 2 == 0 ;
}

//Java
boolean est_pair(int nombre){
        return nombre % 2 == 0 ;
}
```

type de retour
int en C


Client
xyzabc
ssh(client)
pub
Server
ssh(server)
priv
pub

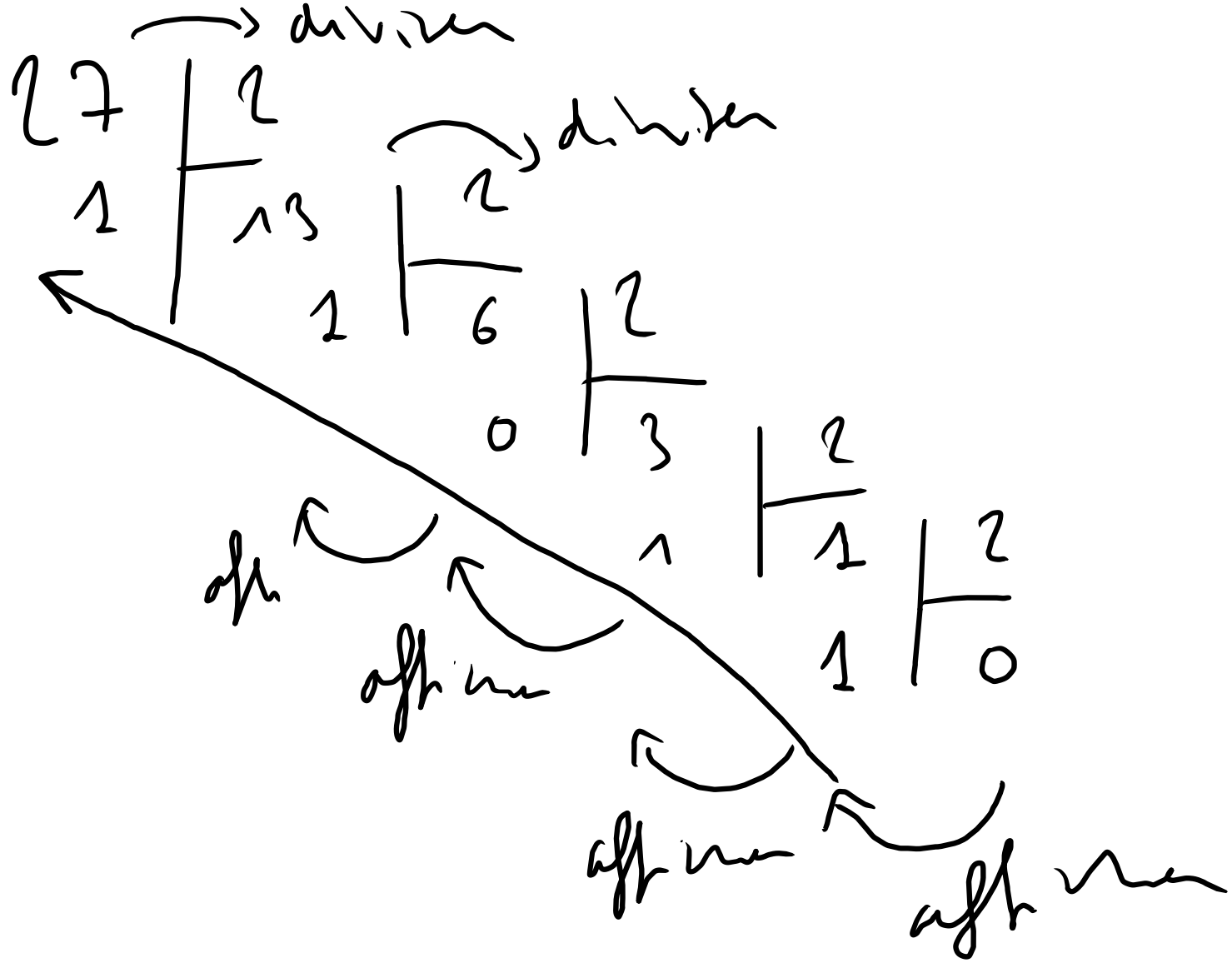
fingerprint de la clé publique du serveur



UI : User Interface

GUI : Graphical User Interface

CLI: Command Line Interface



1^4	1^3	1^2	1^1	1^0
1	0	1	0	1

16 4 1

$(16)_{10} \rightarrow (10000)$

Factorielle

$0! = 1$

$1! = 0! \times 1$

$2! = 1! \times 2$

$5! = 1 \times 2 \times 3 \times 4 \times 5$

$4! = 1 \times 2 \times 3 \times 4$

$10000! = 1 \times 2 \times 3 \times \ldots \times 9999 \times 10000$

Récursivité

```
f(x) = x!

f(x) = f(x-1) * x pour x > 0
f(0) = 1
```

$n! = \underbrace{1 \times \ldots \times (n-1)}_{(n-1)!} \times n$

$n! = (n-1)! \times n$

$(n-1)! = (n-2)! \times (n-1)$

//Itérative

```
Fonction Factorielle(n)
Variable
entier: resultat
Début
     resultat = 1
     Tant que n > 0
     Faire
          resultat <-- resultat * n

     Fin Faire
     retourne resultat
Fin
```

//Récursivité

```
Fonction Factorielle(n)
Début
     Si n > 0 Alors
          retourne Factorielle(n-1) * n
     Sinon
          retourne 1
     Fin Si
Fin
```

Récursivité

$f(x) = x!$

$f(x) = f(x-1) * x \text{ pour } x > 0$
$f(0) = 1$

//Récursivité

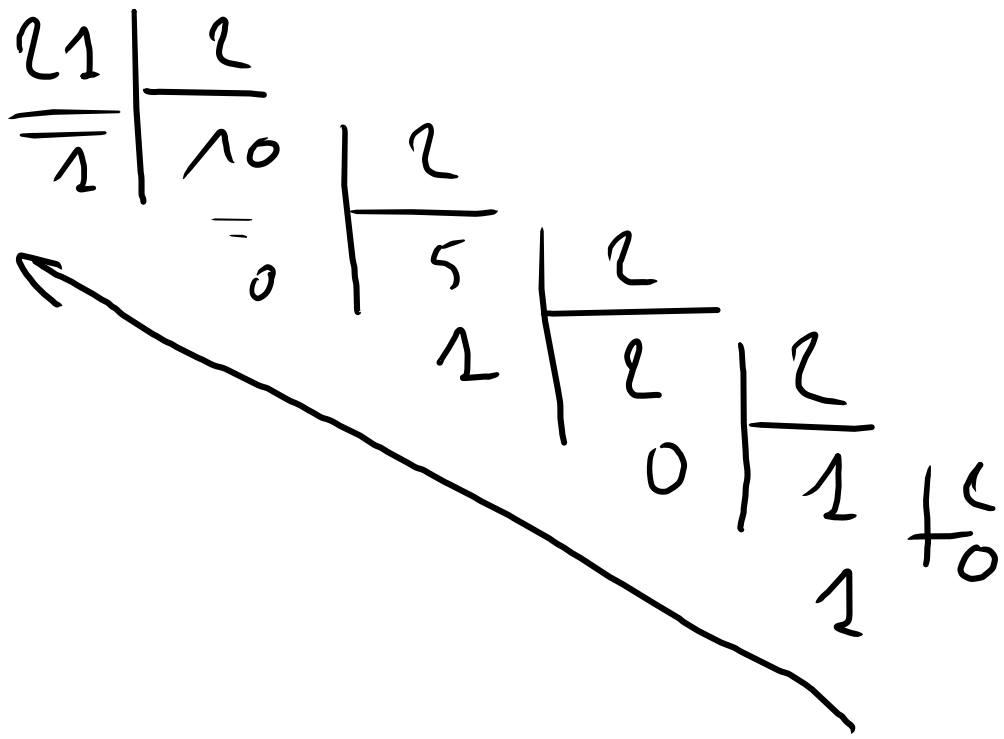
```
Fonction Factorielle(n)
Début
     Si n > 0 Alors
          retourne Factorielle(n-1) * n
     Sinon
          retourne 1
     Fin Si
Fin
```

//Récursivité

```
Fonction Factorielle(n)
Début
     Si n < 0 Alors
          retourne ERREUR
     Fin Si
     Si n == 0 Alors
          retourne 1
     Sinon
          retourne Factorielle(n-1) * n
     Fin Si
Fin
```

$(21)_{10} \rightarrow$ base 2

$(21)_{10} = (10101)_2$



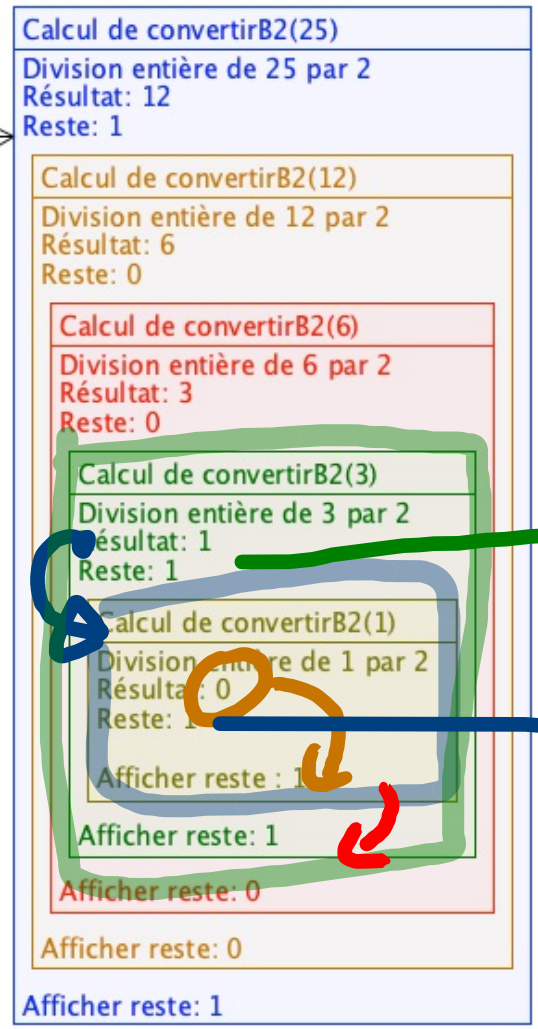
Écrire la fonction convertirB2(n) qui convertit le nombre n exprimé en base 10 en un nombre exprimé en base 2
=> de manière récursive

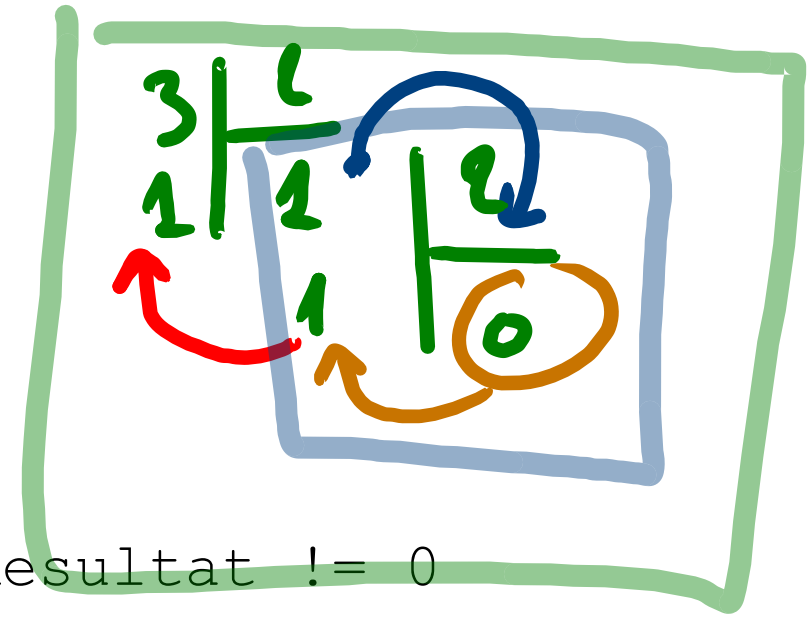
Programme principal
Début

convertirB2(25)

Fin

1 – Fonction convertirB2(n: entier)
2 – Début fonction
3 –
4 –
5 –
6 –
7 –
8 – FIN fonction



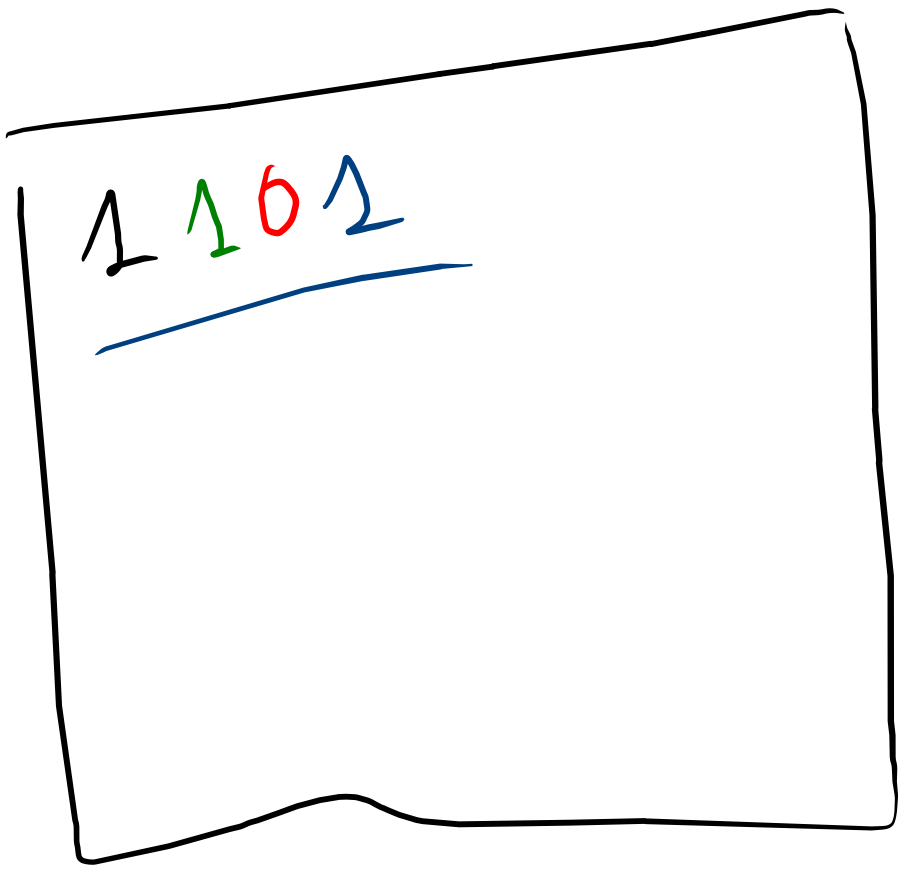


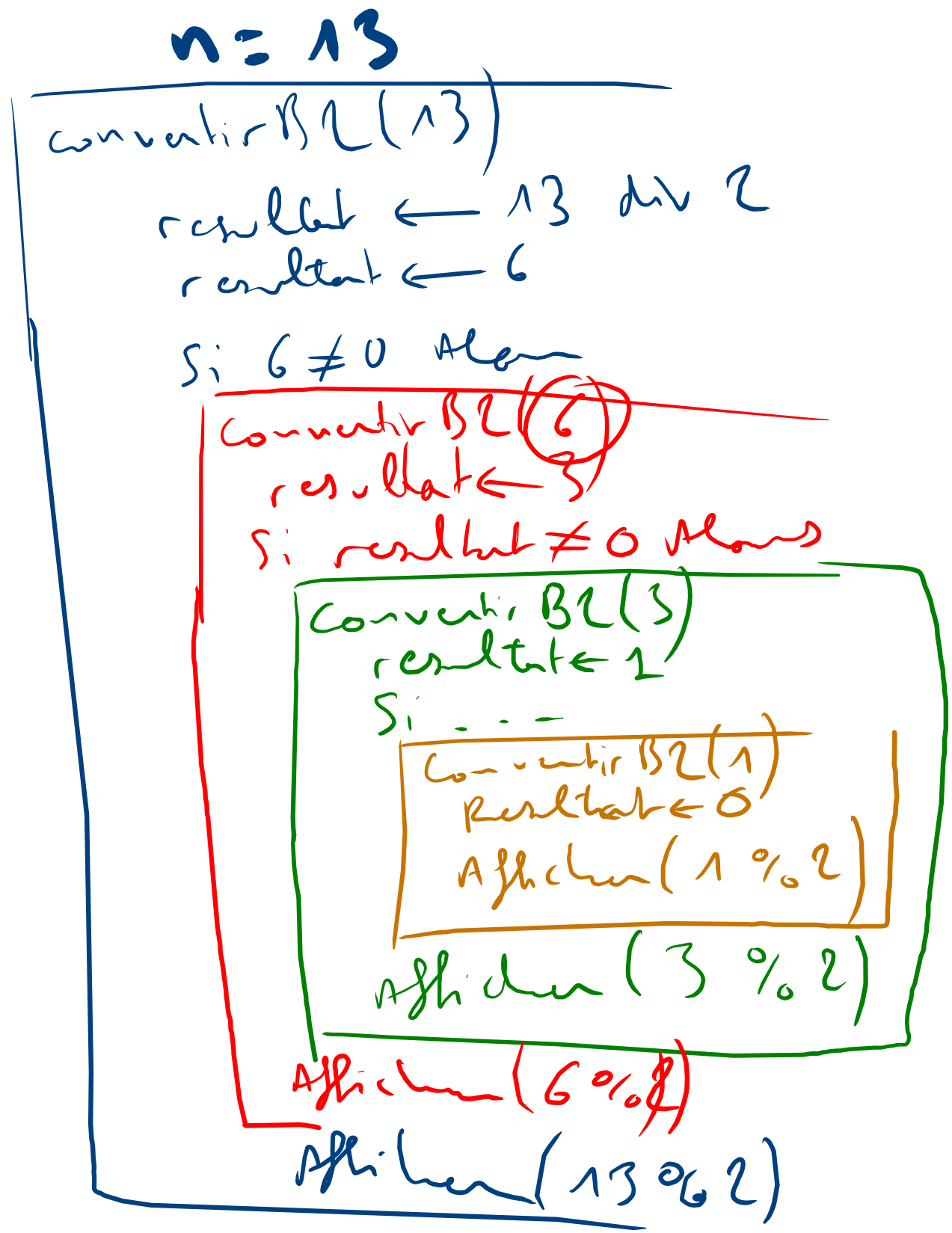
Resultat != 0

Resultat == 0

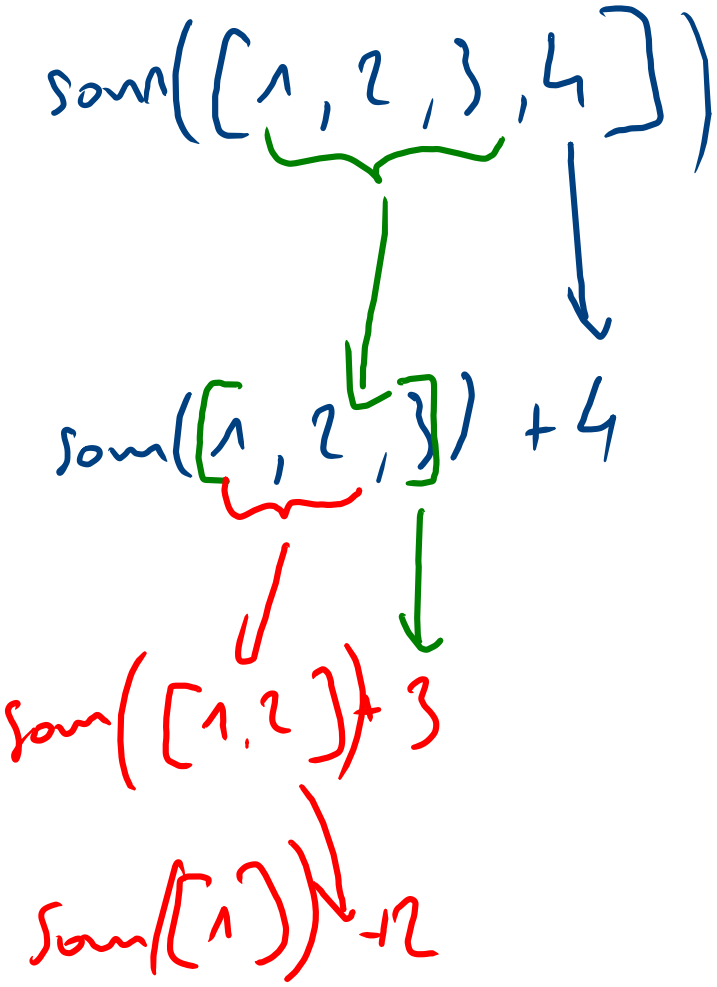
```
Fonction convertirB2(n : entier)
DEBUT
      resultat <-- DIV_ENTIERE(n, 2)
      SI resultat != 0 ALORS
            convertirB2(resultat)
      FIN SI
      AFFICHER(reste(n, 2))
FIN
```

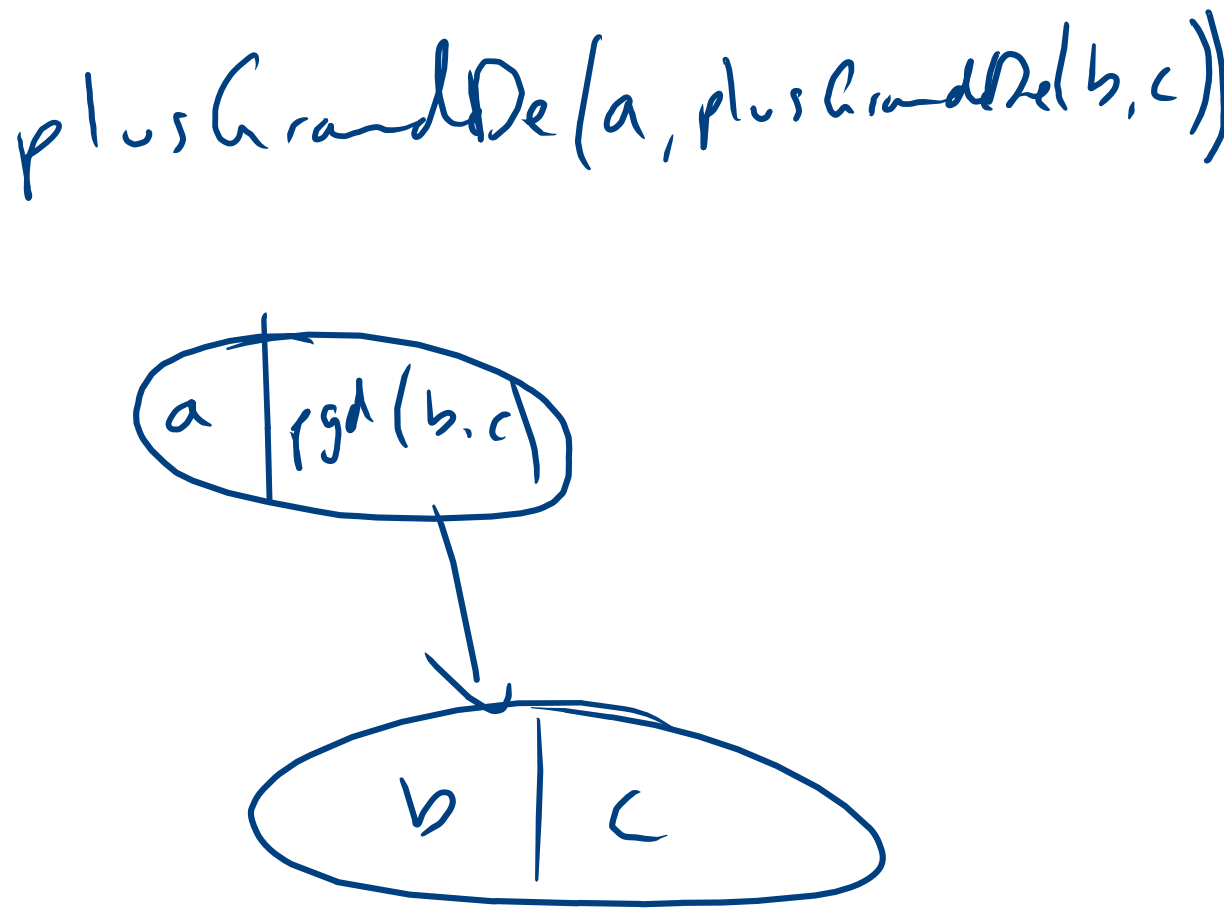
```
Fonction convertirB2(n : entier)
DEBUT
      resultat <-- DIV_ENTIERE(n, 2)
      SI resultat != 0 ALORS
            convertirB2(resultat)
      FIN SI
      AFFICHER(MOD(n , 2))
FIN
```





```
Fonction convertirB2(n : entier)
DEBUT
    resultat <-- DIV_ENTIERE(n, 2)
    SI resultat != 0 ALORS
        retourne convertirB2(resultat) & MOD(n, 2)
    SINON
        retourne MOD(n, 2)
    FIN SI
FIN
```


soma([1,2,3,4])
soma([1,2,3]) + 4
soma([1,2]) + 3
soma([1]) + 2


plusGrandDe(a, plusGrandDe(b,c))
a
pgd(b,c)
b
c

```
CompteARebours (n: entier)
DEBUT
    AFFICHER(n)
    SI n > 0 ALORS
        CompteARebours(n-1)
    FIN SI
FIN
```

```
1 Compter (n: entier)
2 DEBUT
3   SI n > 0 ALORS
4       Compter(n-1)
5       AFFICHER(n)
6   FIN SI
7 FIN
```

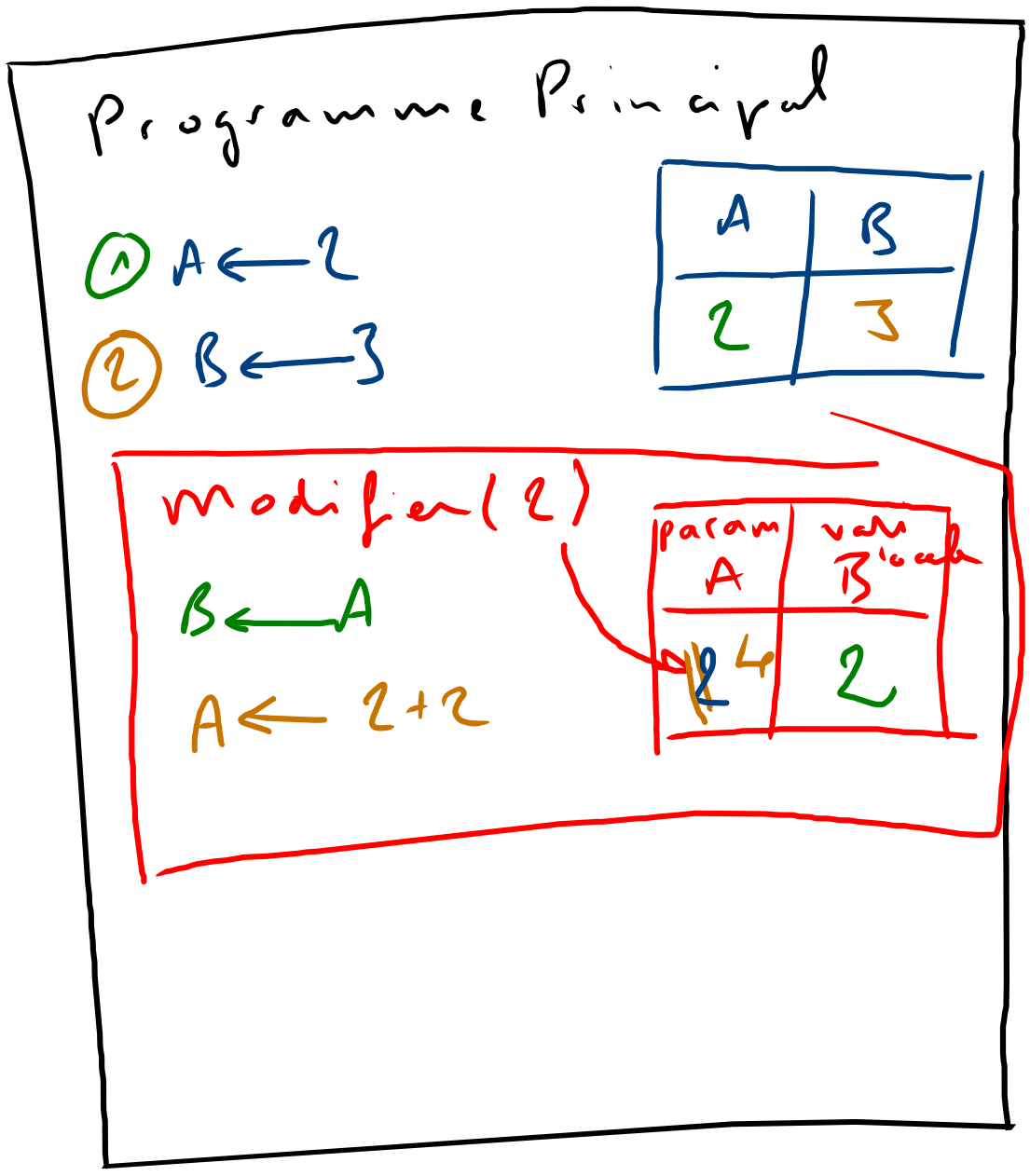
.

Variante 1 : compter de 0

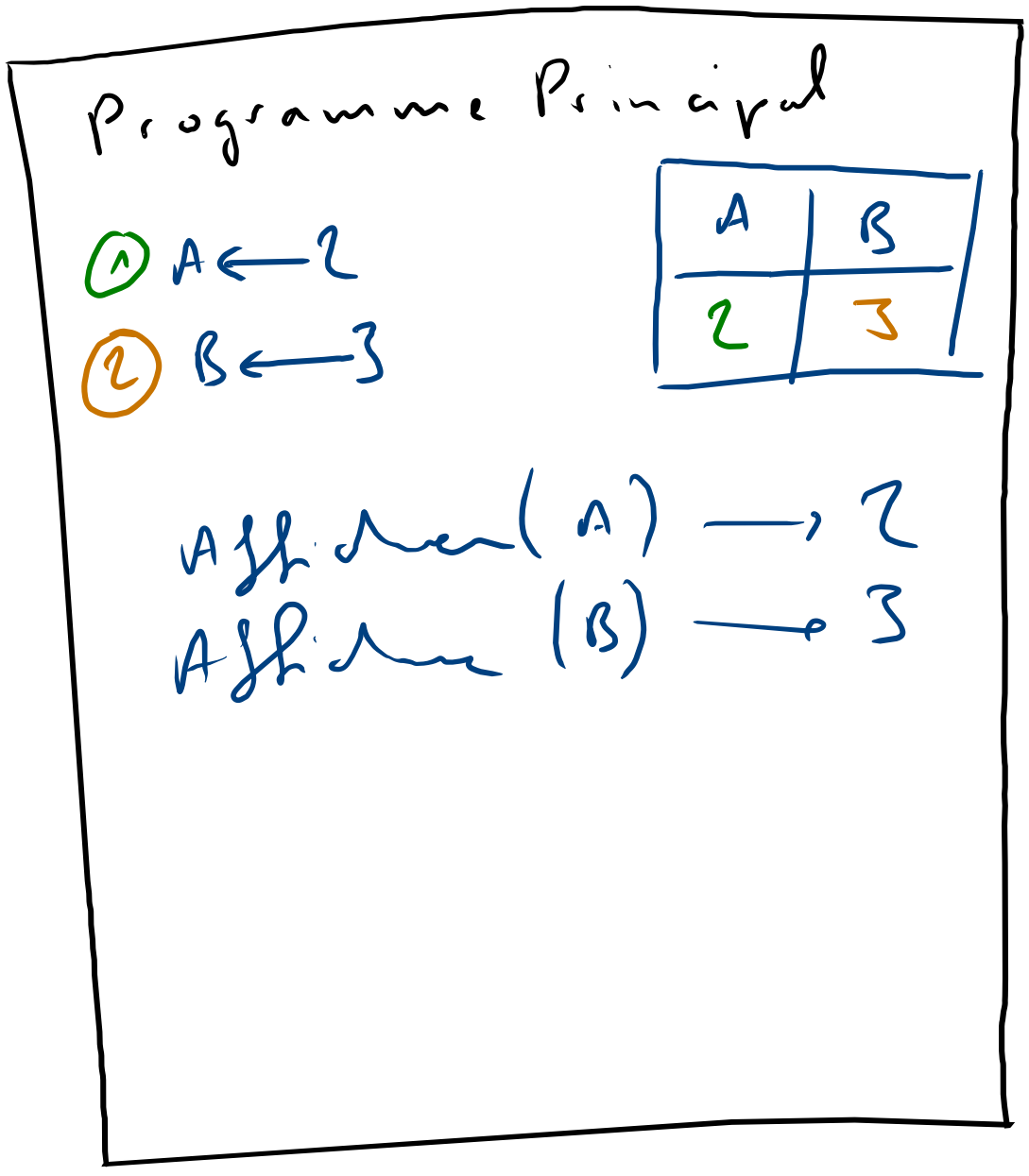
```
1 Compter (n: entier)
2 DEBUT
3   SI n >= 0 ALORS
4       Compter(n-1)
5       AFFICHER(n)
6   FIN SI
7 FIN
```

Variante 2 : compter de 0

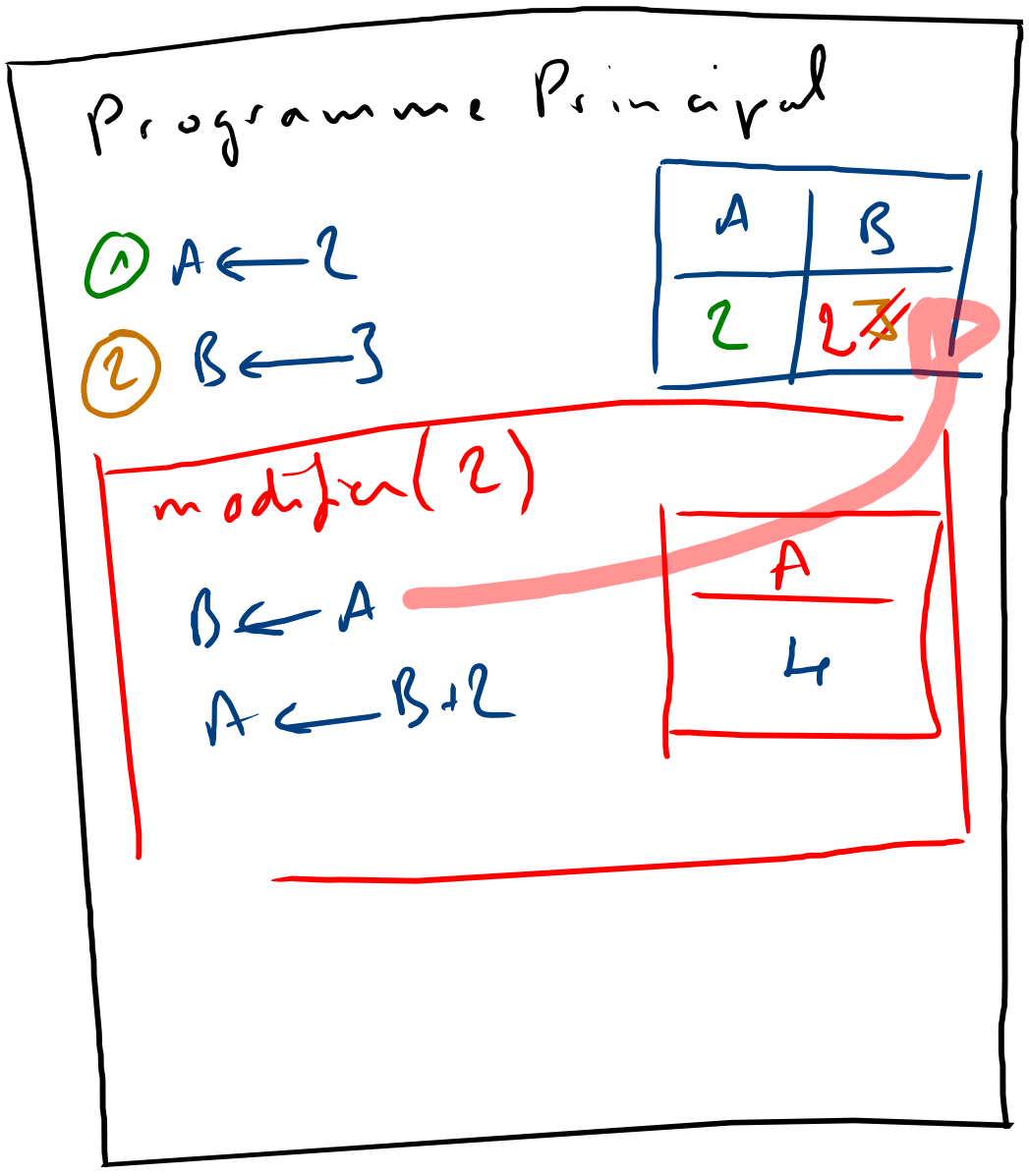
```
1 Compter (n: entier)
2 DEBUT
3   SI n > 0 ALORS
4       Compter(n-1)
5   FIN SI
6   AFFICHER(n)
7 FIN
```



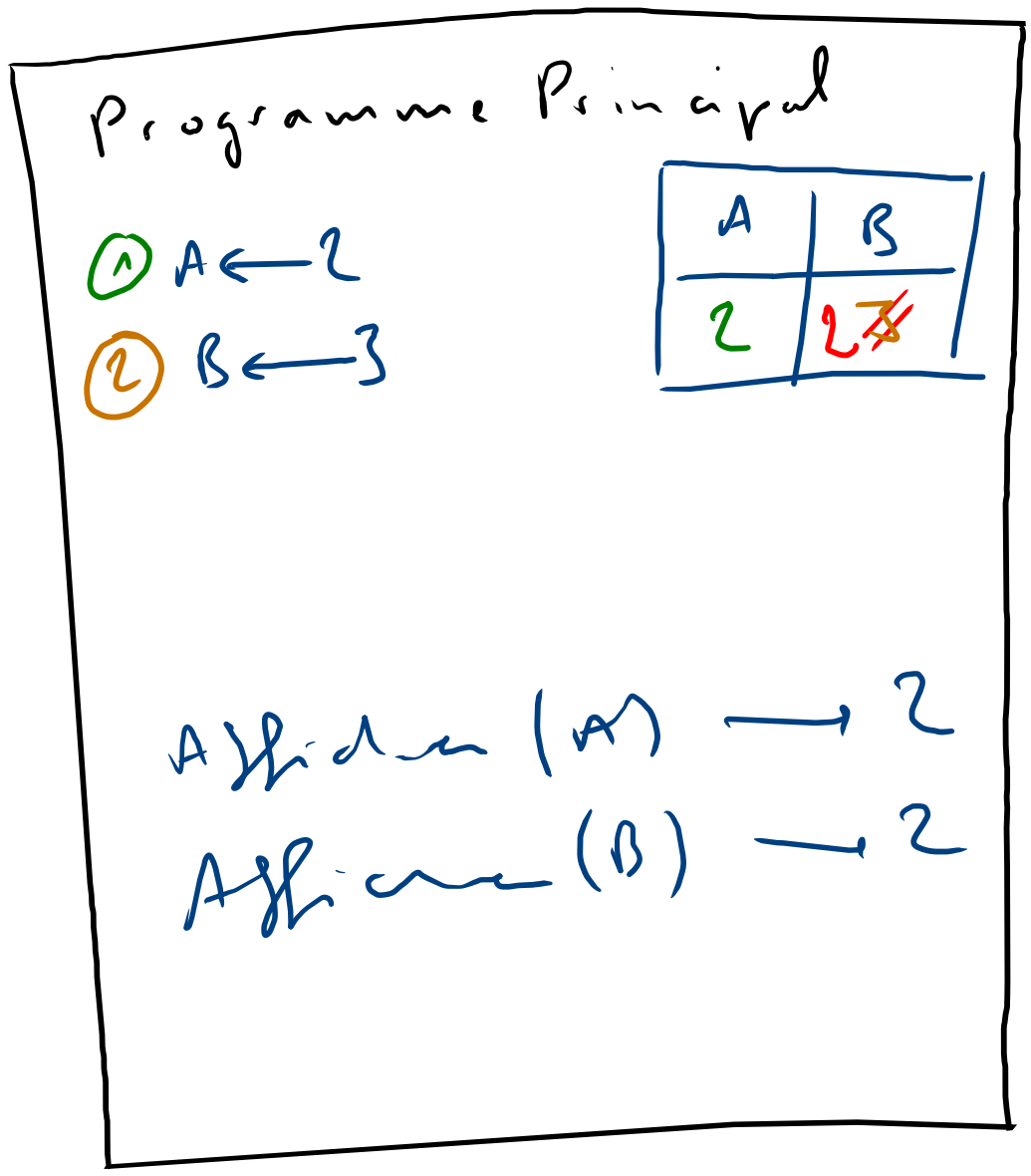

```
Procédure Modifier(A: entier)
Variables
B: entier
DEBUT
      B <-- A
      A <-- B + 2
FIN

Programme principal
Variables
A: entier
B: entier
DEBUT
      A <-- 2
      B <-- 3
      Modifier(A)

      AFFICHER(A)
      AFFICHER(B)
FIN
```



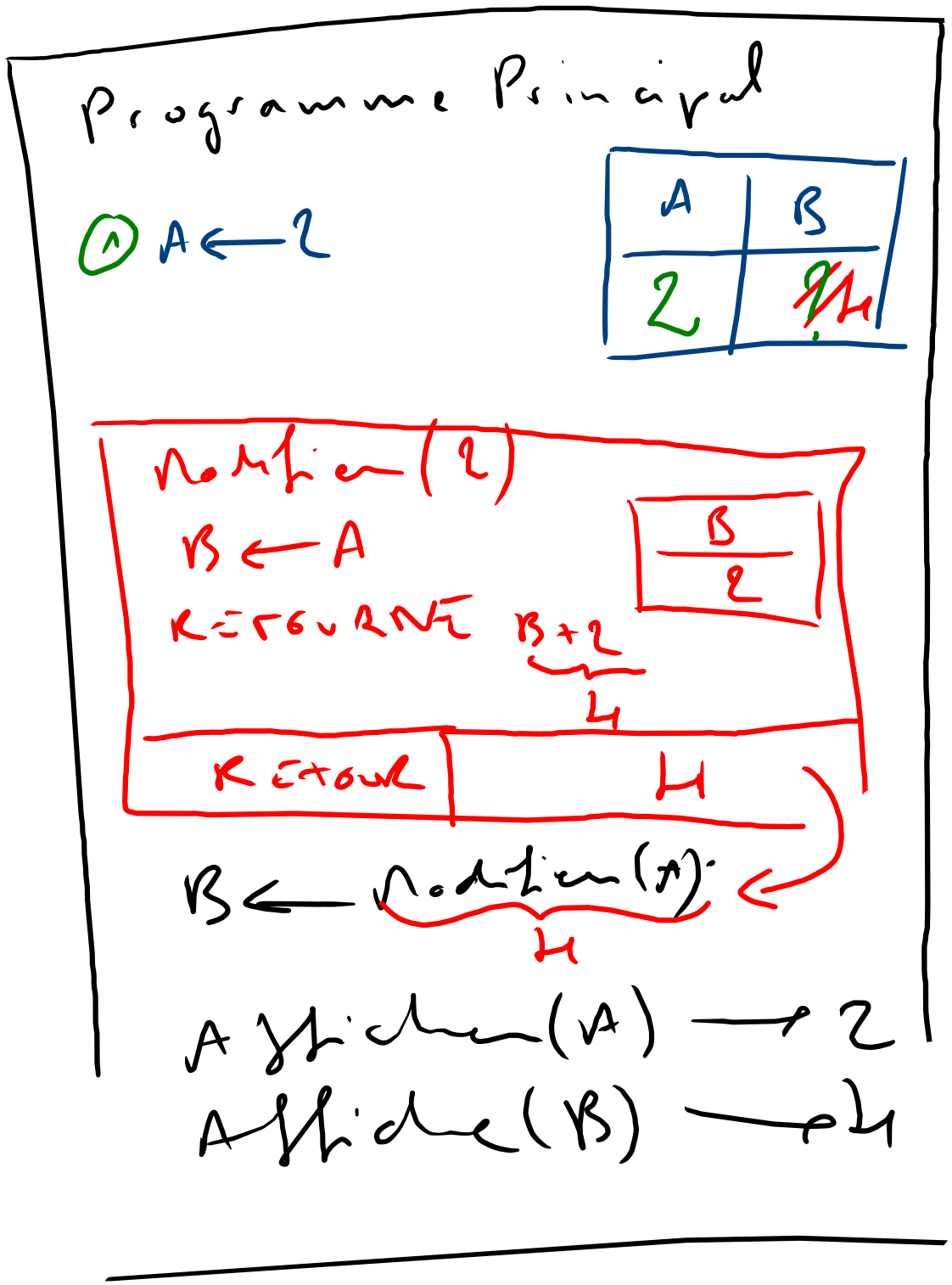

```
Procédure Modifier(A: entier)
Variables
B: entier
DEBUT
      B <-- A
      A <-- B + 2
FIN

Programme principal
Variables
A: entier
B: entier
DEBUT
      A <-- 2
      B <-- 3
      Modifier(A)

      AFFICHER(A)
      AFFICHER(B)
FIN
```




```
//AVEC VARIABLE GLOBALE
Procédure Modifier(A: entier)
Variables

DEBUT
      B <-- A
      A <-- B + 2
FIN

Programme principal
Variables
A: entier
GLOBALE B: entier
DEBUT
      A <-- 2
      B <-- 3
      Modifier(A)

      AFFICHER(A)
      AFFICHER(B)
FIN
```

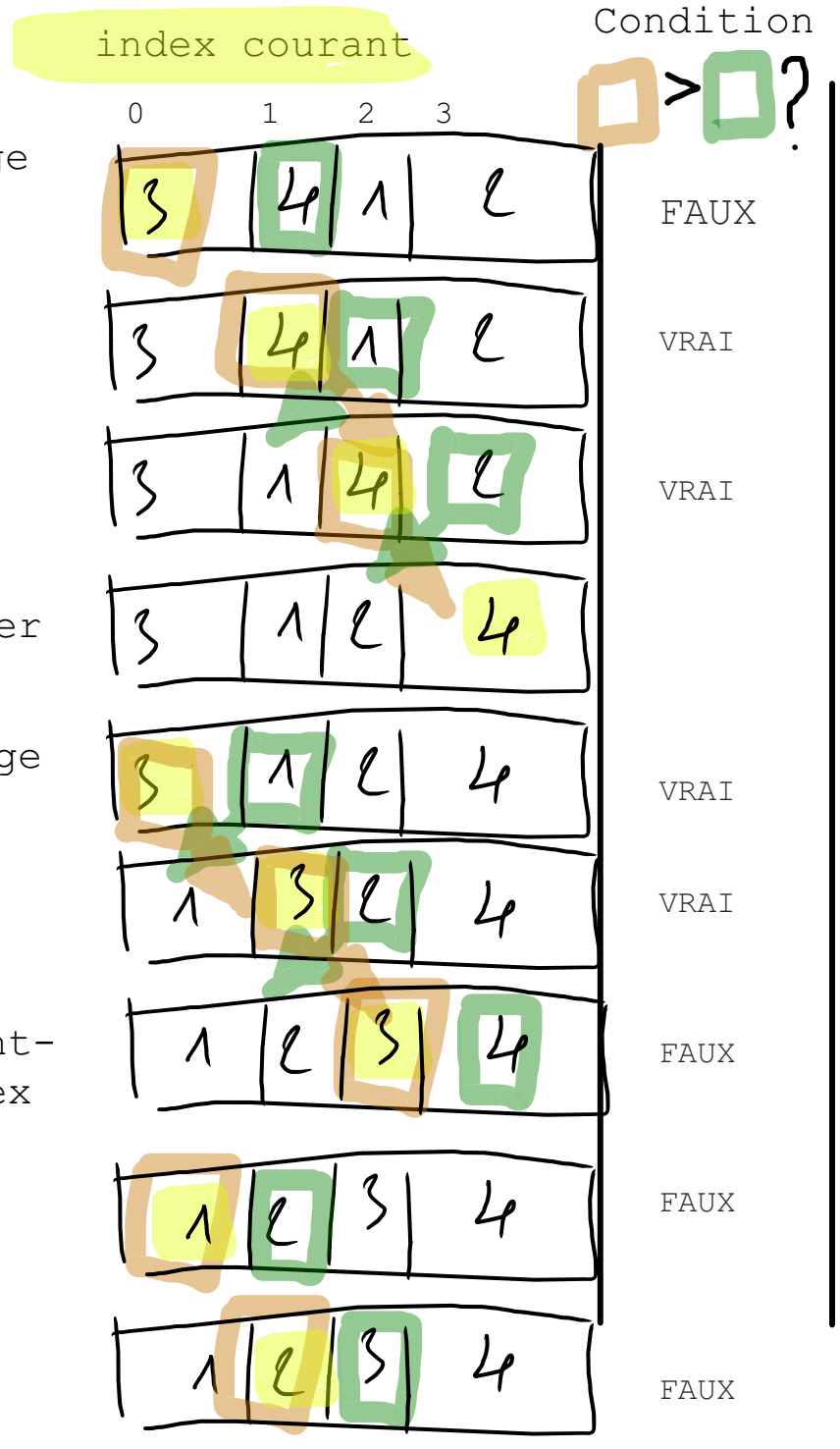



```
//AVEC VARIABLE GLOBALE
Procédure Modifier(A: entier)
Variables

DEBUT
      B <-- A
      A <-- B + 2
FIN

Programme principal
Variables
A: entier
GLOBALE B: entier
DEBUT
      A <-- 2
      B <-- 3
      Modifier(A)

      AFFICHER(A)
      AFFICHER(B)
FIN
```

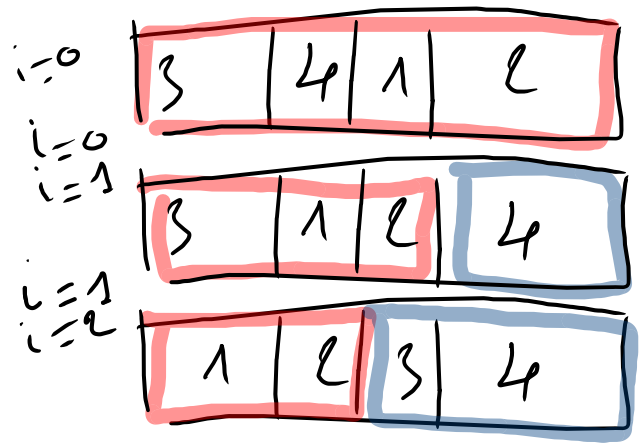


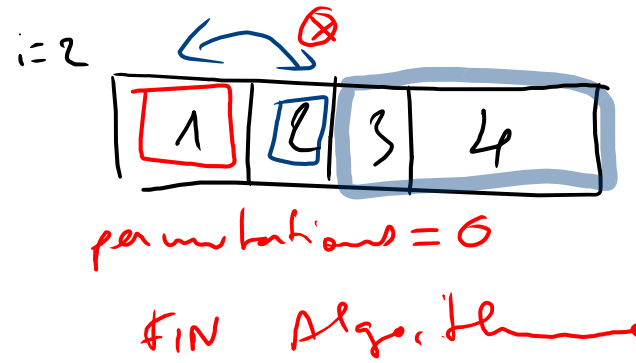

```
//AVEC VARIABLE GLOBALE
FONCTION Modifier(A: entier)
Variables
B: entier
DEBUT
      B <-- A
      RETOURNE B + 2
FIN

Programme principal
Variables
A: entier
B: entier
DEBUT
      A <-- 2
      B <-- Modifier(A)

      AFFICHER(A)
      AFFICHER(B)
FIN
```

Tri à bulle



Fin si aucune permutation

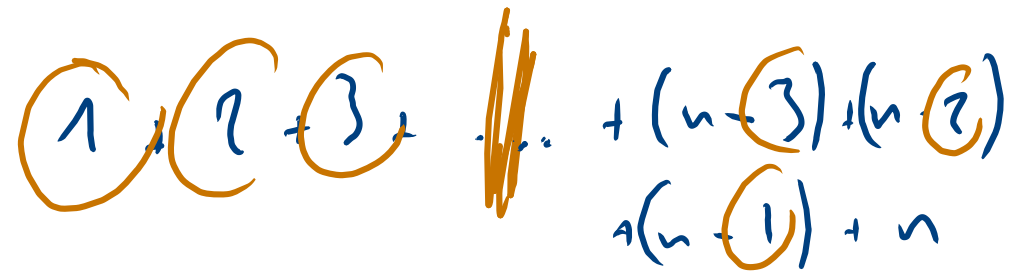


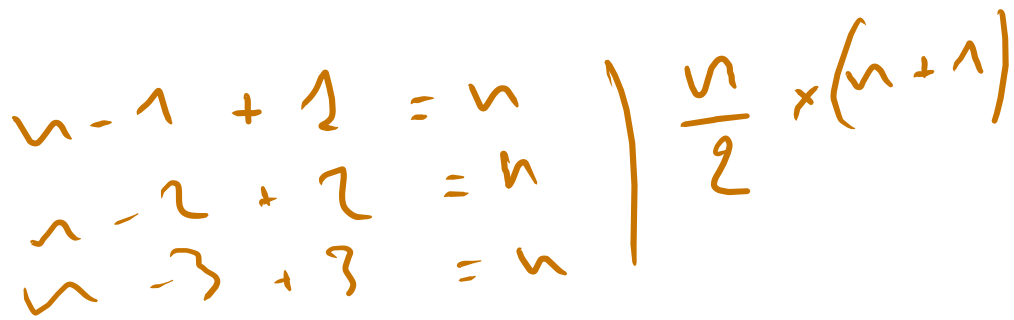


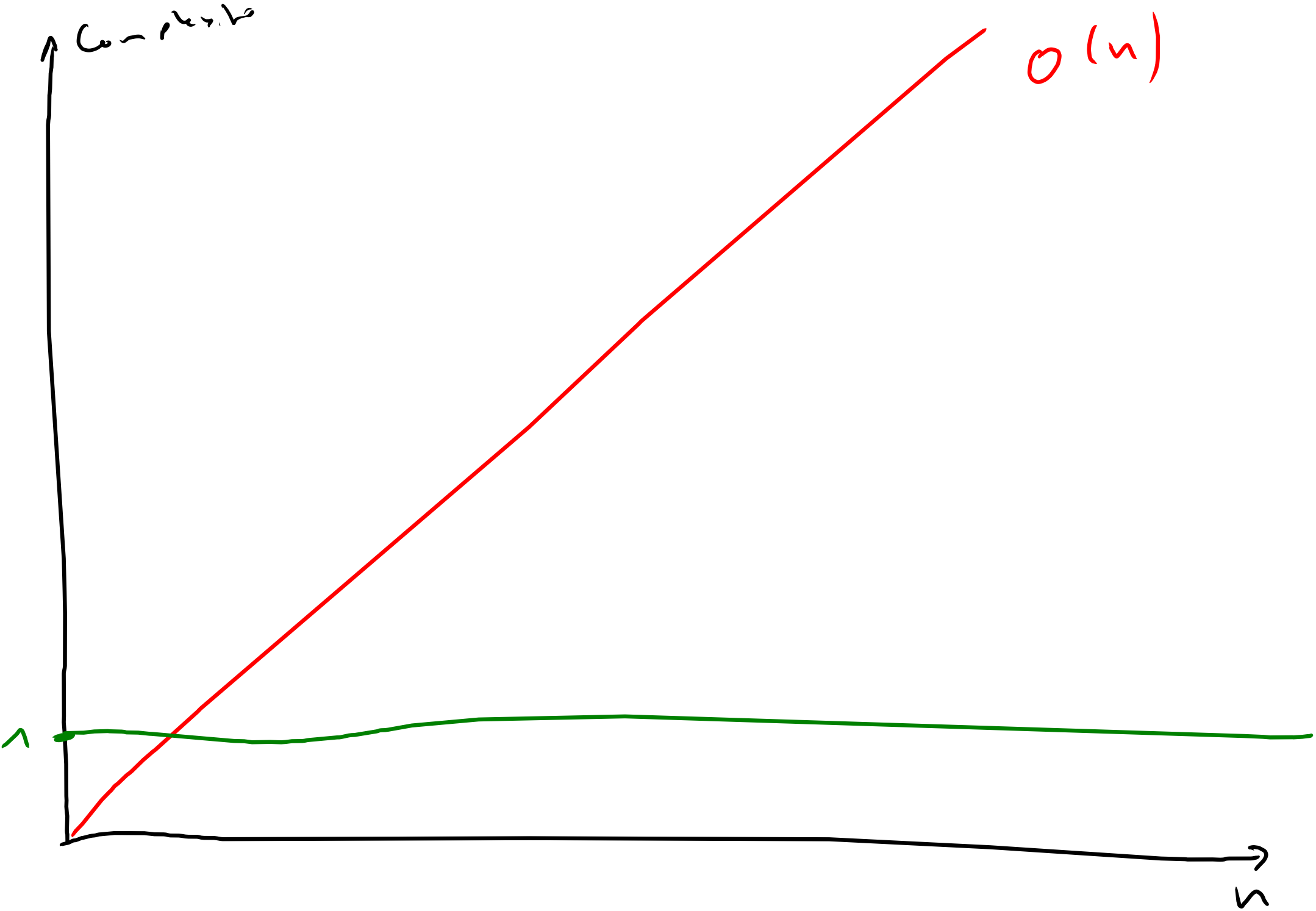
```python
n = 3
output = getSum(n)
print(output)
# imprime : 6  car 1 + 2 + 3 = 6

def getSum(n):
    sum = 0
    for number in range(1, n + 1):
        sum += number
    return sum
```

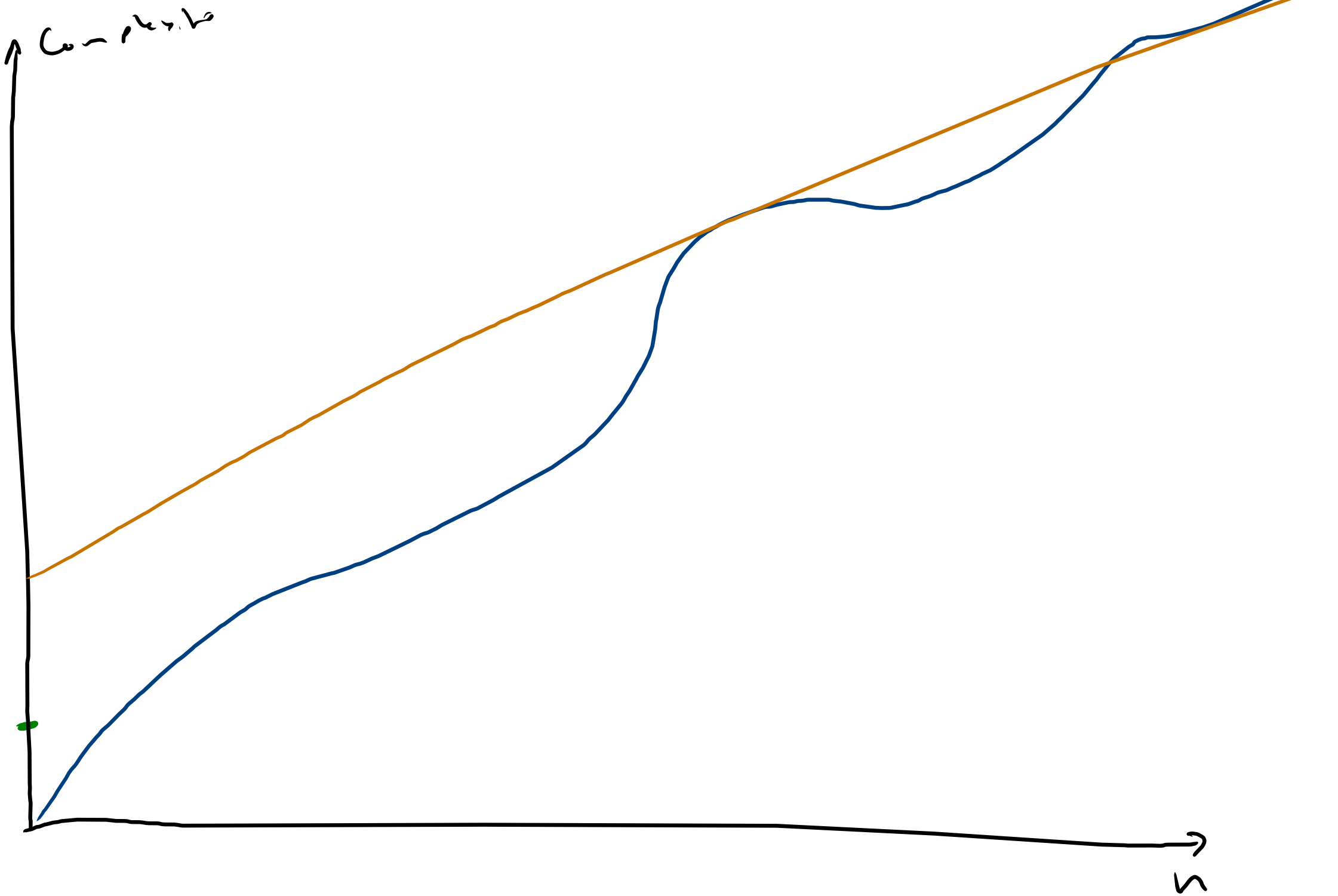
```python
def getSum(n):
    return n * (n + 1) / 2
```

Salut ! Ta solution avait une tendance linéaire O(n) sauf que l'input était énorme. J'ai trouvé une solution constante O(1). Peu importe l'input, ça sera rapide maintenant.






O(n)
1
n


n

Complexité O(1)

```
Programme
Variables
    age: entier
Début
    AFFICHER("Indiquer votre âge:")
    age <-- LIRE()
Fin
```

```
Programme
Variables
    age: entier
    genre: texte
    nom: texte
    prenom: texte
Début
    AFFICHER("Indiquer votre âge:")
    age <-- LIRE()
    AFFICHER("Indiquer votre genre:")
    age <-- LIRE()
    AFFICHER("Indiquer votre nom:")
    nom <-- LIRE()
    AFFICHER("Indiquer votre prenom:")
    prenom <-- LIRE()
Fin
```

Complexité linéaire O(n)

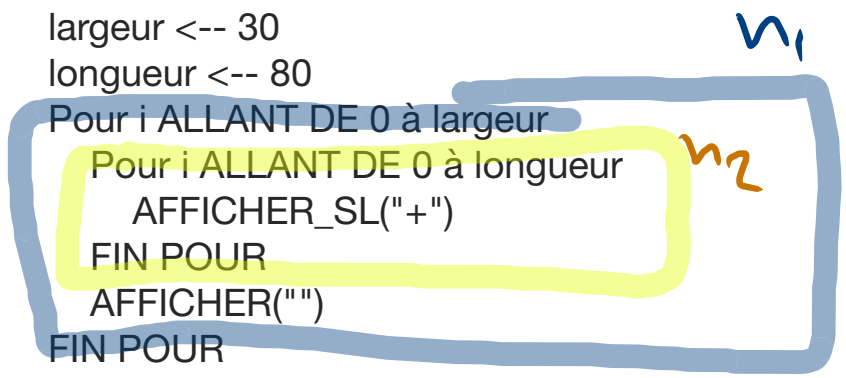
```
Tableau Tableau[14] en Numérique
Variables Moyenne, Somme en Numérique
Début
Somme <-- 0
Pour i <-- 0 à 13
  Somme <-- Somme + Tableau[i]
i Suivant
Moyenne <-- Somme / 14
Fin
```

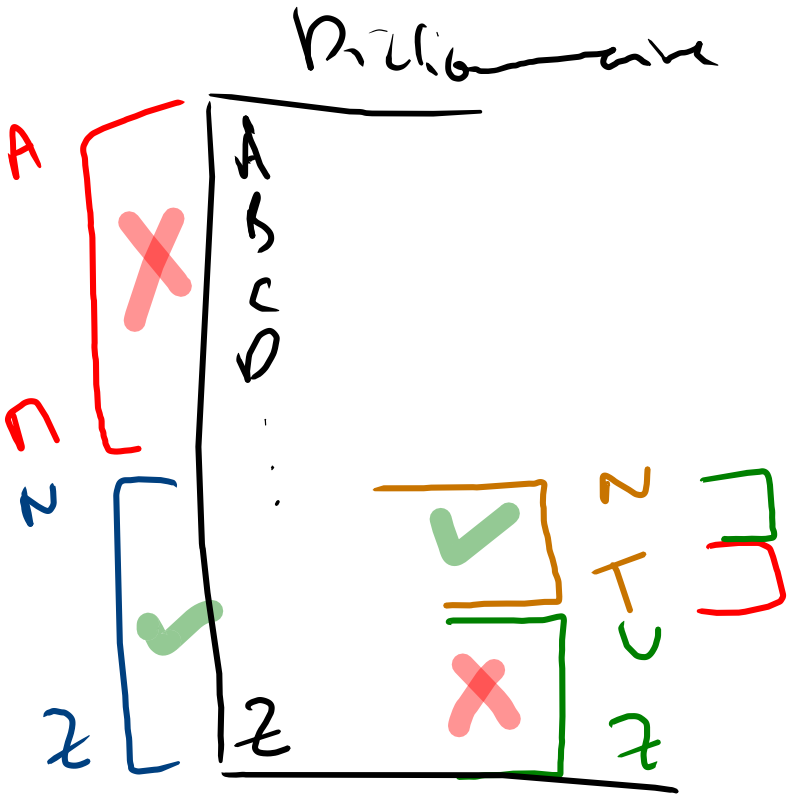
O(2n) = O(n)

```
Tableau Note[11] en Numérique
Variables Moy, Som en Numérique
Début
Pour i <-- 0 à 11
  Ecrire "Entrez la note n°", i
  Lire Note[i]
i Suivant
Som <-- 0
Pour i <-- 0 à 11
  Som <-- Som + Note[i]
i Suivant
Moy <-- Som / 12
Fin
```

`O(n^2) = O(n au carré) complexité quadratique`

```
Variables
largeur, longueur : entier
Début
    largeur <-- 30
    longueur <-- 80
    Pour i ALLANT DE 0 à largeur
        Pour i ALLANT DE 0 à longueur
            AFFICHER_SL("+")
        FIN POUR
        AFFICHER("")
    FIN POUR
Fin
```

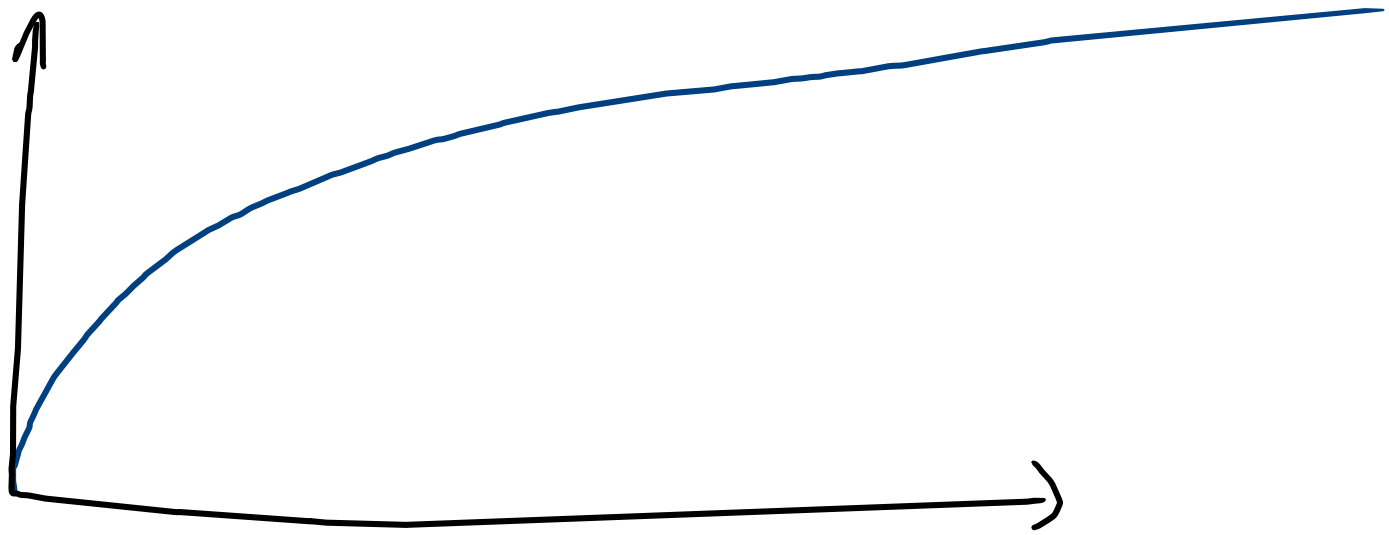




[1 100]

~~Quadratique~~

Logarithmique O(log n)



Déterminer la complexité:

- du tri à bulle
- du tri par insertion

```
Enregistrement nomEnregistrement
    nomChampsA: typeChampsA
    nomChampsB: typeChampsB
    ...
    nomChampsX: typeChampsX
FinEnregistrement
```

Variables
    nomVariable: nomEnregistrement
    origine: Point
    etudiant: Fiche
FinVariables

```
origine.x = 0
origine->x = 0

valeur = origine.x + 2
```

1. Définir un enregistrement destiné à décrire un étudiant à l'aide de son nom, son prénom et sa moyenne.

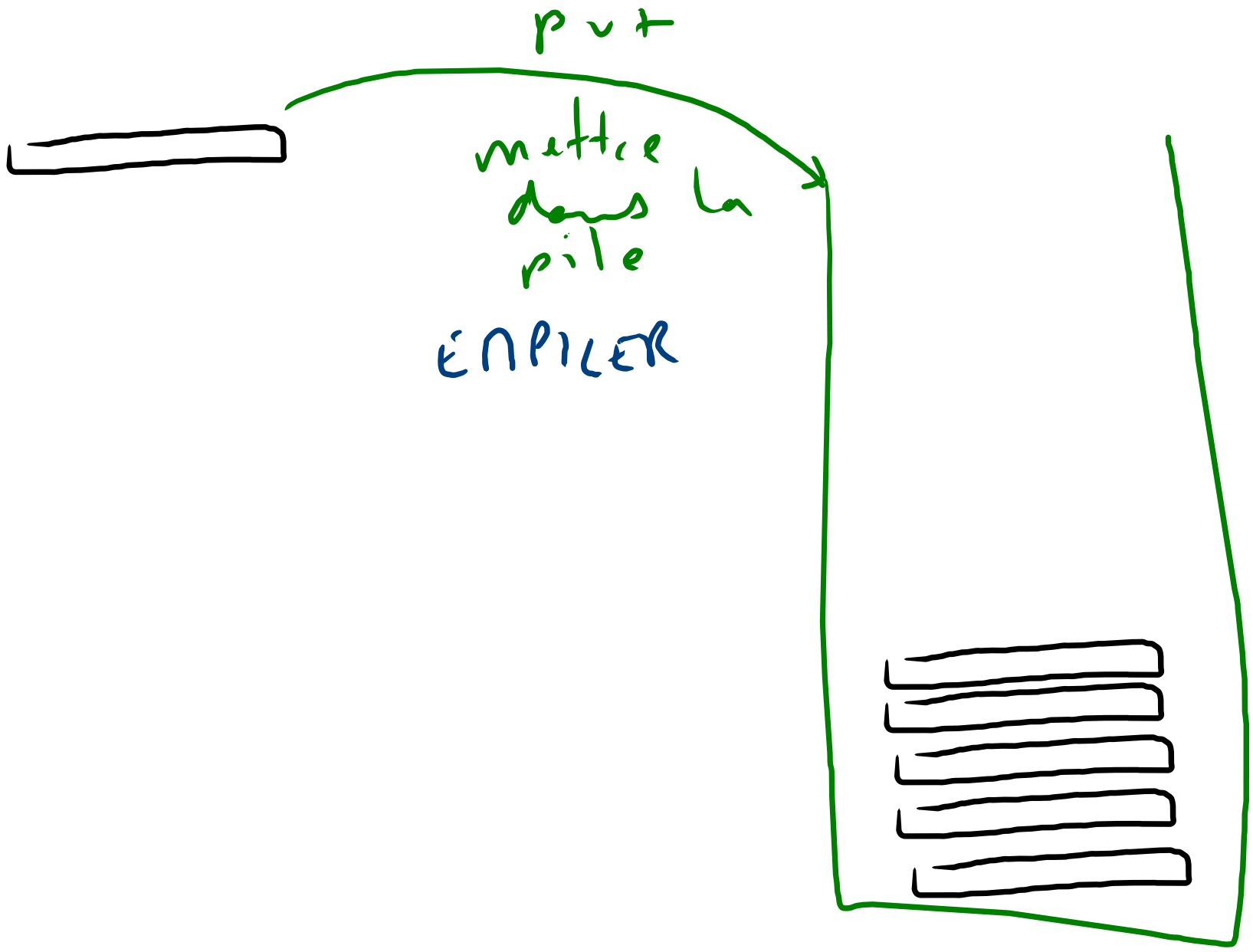
2. Écrire un algorithme permettant de calculer et d'afficher les moyennes d'un groupe de 20 étudiants. Pour cela on procédera comme suit :

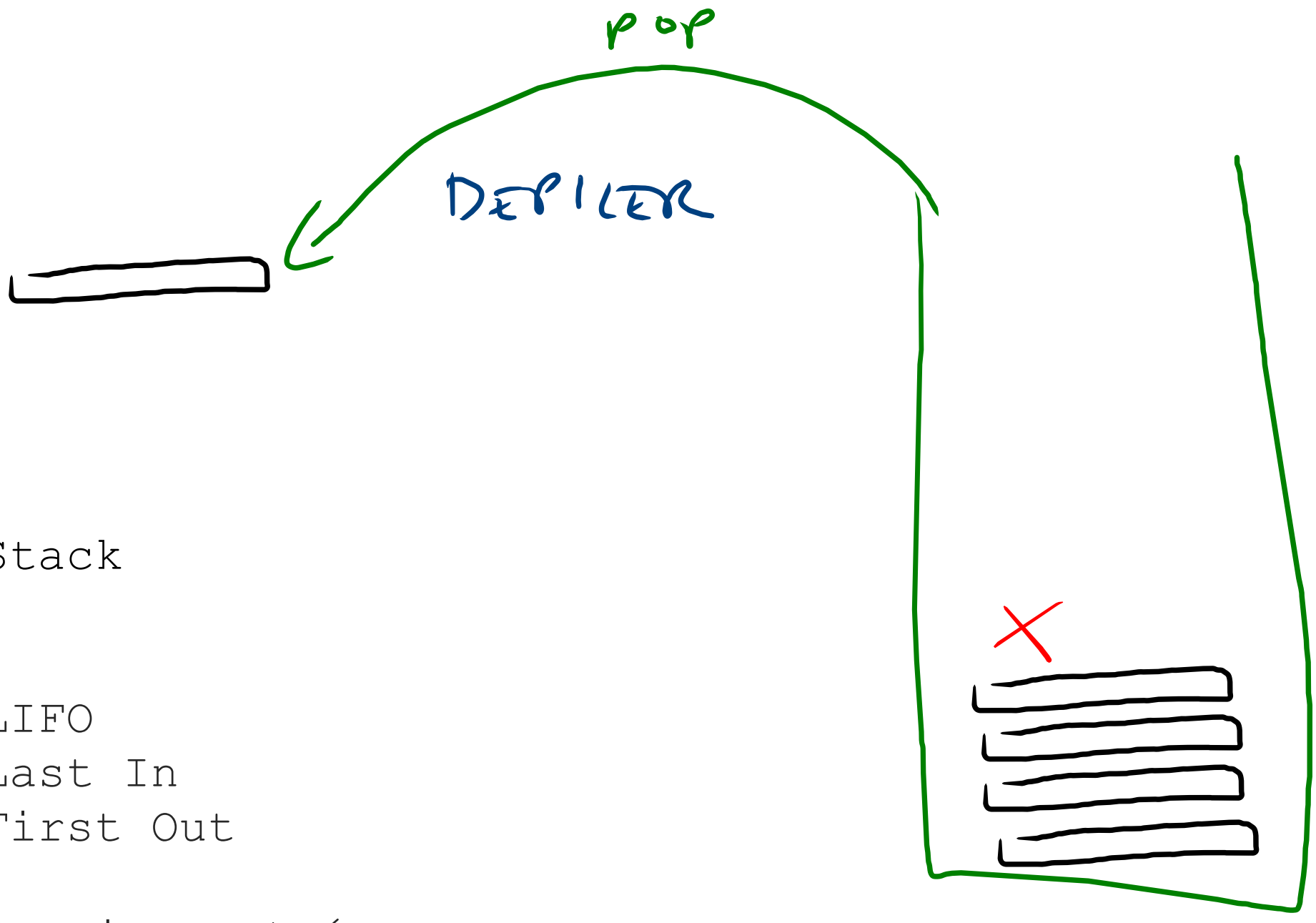
    1. l'utilisateur saisit le nombre de notes nbnotes dont on calculera la moyenne,
    2. pour chaque étudiant, l'utilisateur saisit les nbnotes qu'il a obtenues, ensuite l'algorithme calcule la moyenne de cet étudiant,
    3. à la fin l'algorithme affiche pour chaque étudiant son nom, son prénom et sa moyenne.

3. Reprendre le même exercice en triant les étudiants par moyenne.

```
Enregistrement Etudiant
    nom: texte
    prenom: texte
    moyenne: flottant
FinEnregistrement
```

```
Variables
    etudiants : tableau de 20 étudiants
    nbnotes, e, i: entier
    somme: flottant
    note: flottant
FinVariables
Debut
    Lire(nbnotes)
    Pour e allant de 1 à 20
        somme <- 0
        Ecrire("Etudiant: " & etudiants[e].nom
            & " " & etudiants[e].prenom)

        Pour i allant de 1 à nbnotes
            Ecrire("Saisir note:")
            Lire(note)
            somme <- somme + note
        FinPour
        etudiants[e].moyenne <- somme / nbnote
    FinPour
Fin
```


put
mettre dans la pile
EMPILER

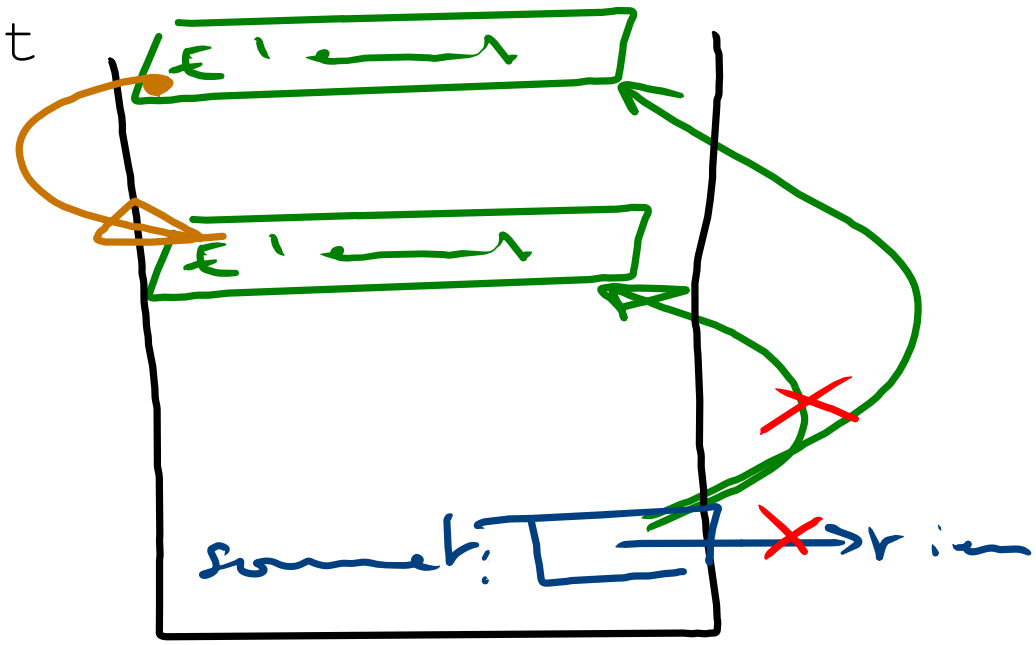


Stack

LIFO
Last In
First Out

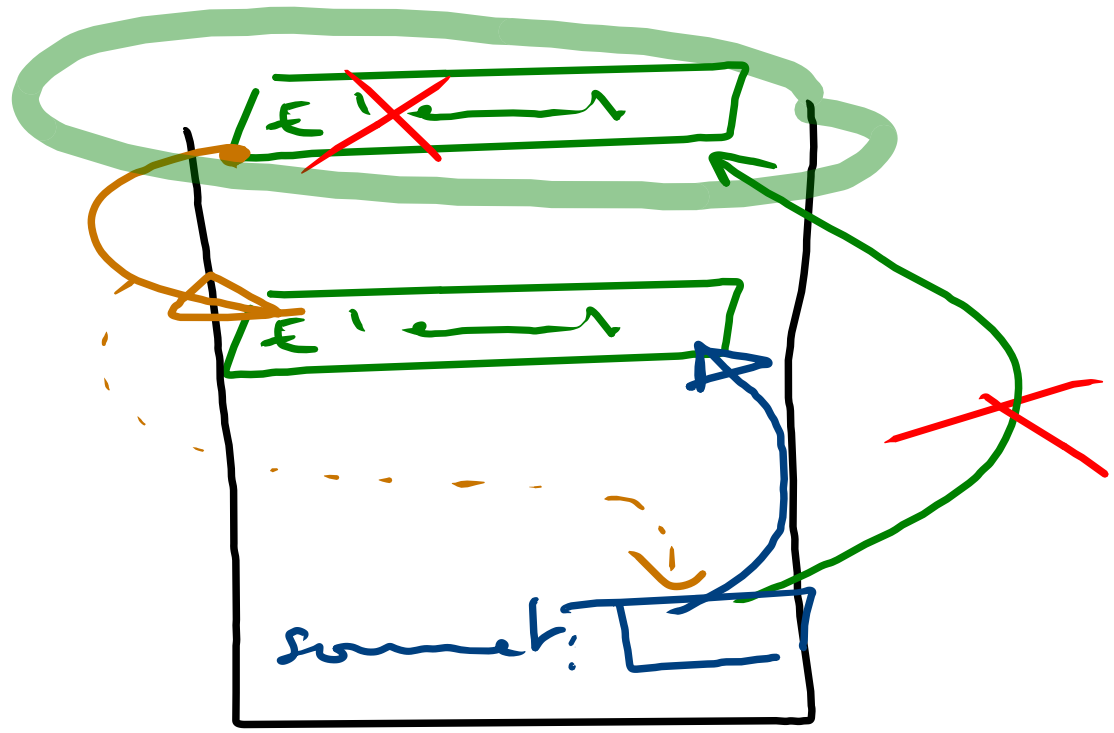
Dernier entré
Premier sorti

```
Enregistrement Element
    valeur: Texte
    prochain: Element
FinEnregistrement

Enregistrement Pile
    sommet: Element
FinEnregistrement
```

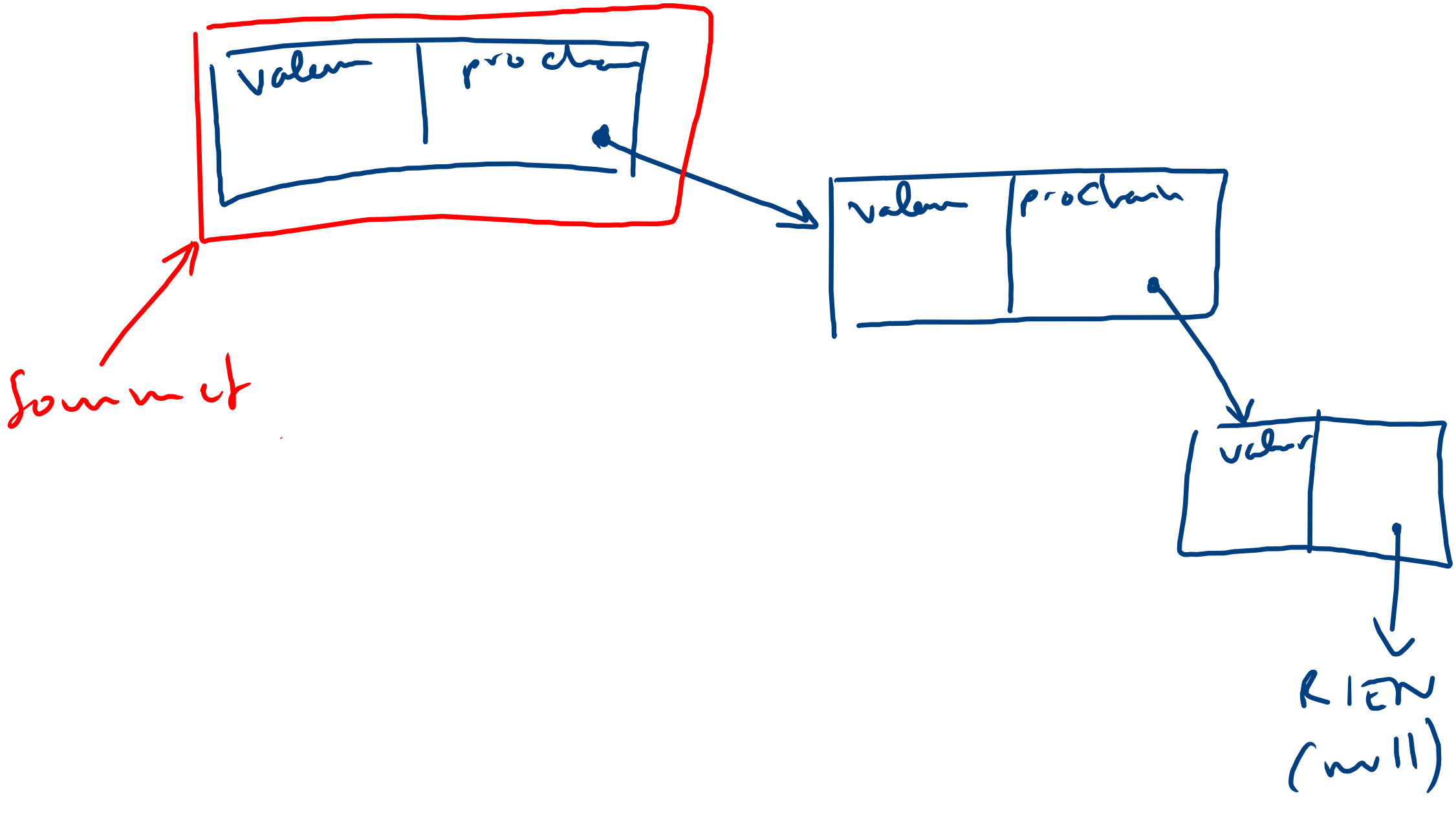


```
Pile creerPile()
Element empiler(valeur:Texte, pile: Pile)
booléen estVide(pile: Pile)
Element depiler(pile: Pile)
Element sommet(pile: Pile)
```

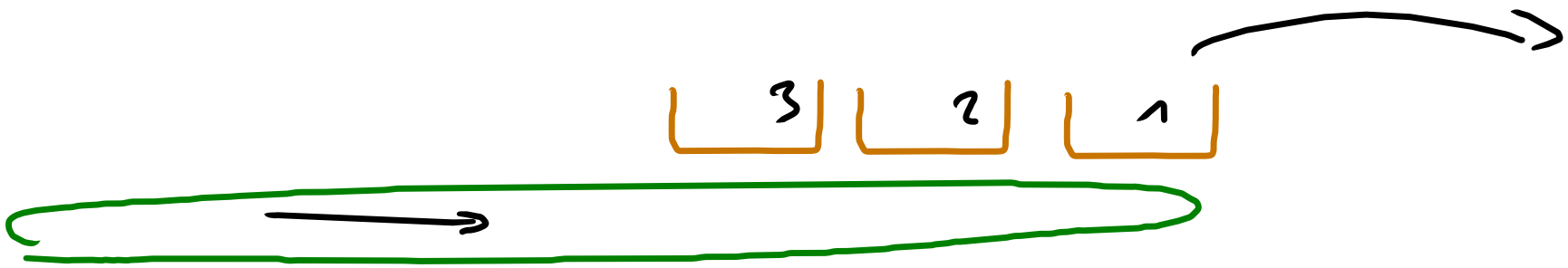
```
Enregistrement Element
    valeur: Texte
    prochain: Element
FinEnregistrement

Enregistrement Pile
    sommet: Element
FinEnregistrement
```



```
Pile creerPile()
Element empiler(valeur:Texte, pile: Pile)
booléen estVide(pile: Pile)
Element depiler(pile: Pile)
Element sommet(pile: Pile)
```


valeur
prochain
valeur
prochain
sommet
valeur
RIEN
(null)

First In First Out : FIFO
Queue = File d'attente
File




```
Enregistrement Element
    valeur: Texte
    prochain: Element
FinEnregistrement

Enregistrement File
    suivant: Element
FinEnregistrement
```

```
File creerFile()
Element enfiler(valeur:Texte, file: File)
booléen estVide(file: File)
Element defiler(file: File)
Element sommet(file: File)
```