Tri Selection:

procedure Selection(n: entier, @ t₀ :tab):

début

pour i de 1 a n-1 fois

j ← i

tmp ← t[i]

tant que j > 1 et tmp < t[j-1] fois

t[j] ← t[j-1]

j ← j-1

fin tant que

t[j] ← tmp

Fin pour

Fin

Var	type
i	entier
j	entier
tmp	entier ou réel !

Tri a bulle:

procedure bubble(n: entier, @ t:tab):

Debut

pour i de 0 à n-2 fois:

Si t[i] > t[i+1] alors

tmp ← t[i]

t[i] ← t[i+1]

t[i+1] ← tmp

bubble(n, t)

Fin Si

Fin pour

Fin

Var	type
i	entier
tmp	entier ou réel.
bubble	procedure

Tri Shelle:

Procedure Shell (n: entier, @ t: tab):

Debut:

Pas ← 1

tant que pas × 3 + 1 < n faire
    Pas ← Pas × 3 + 1
Fin tant que.

Repeter:
Pour i de Pas a n - 1 faire

tmp ← t[i]

j ← i

tant que j ⩾ 1 et t[j - Pas] < tmp faire

t[j] ← t[j - Pas]

j ← j - Pas

Fin tant que

t[j] ← tmp

Fin pour

Pas ← Pas div 3

Jusqu'a ce (Pas < 1)

tmp j ⩾ j   entier
Pas   entier

Fin

project : Mathcomplex:

Cartésienne :

Def func calc : $(2-2i)^2$

$a^2 + 2ab + b^2$

function calc (reg: String) : String :

Begin

// séparate :

13/12/24:

Correction devoir Synth n°1:

P.P

Algorithme point-equilibre &

Debut

    Saisir (l, c)
    fillmat (l, c, m)
    src ← "equilibre.dat"
    remplirf (m, src, l, c)

    src₁ ← "mon_Somme.txt"
    remplir St (src, src₁, m)

Fin

TDO:

Nom	type
l, c	entier
src, src₁	chaîne
m	mat

Fonction (m: mat, l: entier, c: entier): booleen;

Debut
    i ← -1
    Repeter
        i ← i+1
        chek ← som[l, i] ≠ m[l, c]
    Jusqu'à chek = faux ou i = c-1.
    Retourner chek.
Fin

TDO:
i: entier
chek: bool

Recursivités :

fonction fact (m: entier) : entier

  Début
    Si m=0 alors f←1
    Sinon
        f←1
        pour i de 1 à (pos-1) faire
            f← f×i

    fin Si pour
    retourner f

  Fin

---

fonction rec (m: entier) : entier
  Début
    Si m=0 alors
        retourner 1
    Sinon
        retourner m × rec (m-1)
    Fin Si
  Fin

---

exemple 2 :
en calcule La somme de m premier entier positive de 2 manieres ;

fonction Som (m: entier) : entier
  Début
    S←0
    pour i de 0 à n faire
        S←S+i
    Fin pour
    Retourner S
  Fin

---

fonction Som (m: entier) : entier
  Début
    Si m=0 alors
        retourner 0
    Sinon
        retourner m + Som (m-1)
    Fin Si
  Fin

Act : ecrire une fonction recursive permettant de effectuer
la multiplication de 2 entier > 0 notter A et B,
en utilisant uniquement L'addition entière.

en effet A×B = A× R ..... B×

```
Fonction Rec(a : entier, b : entier) : entier
    Début
        S ← 1
        Pour i de 1 a b faire
            S ← S*a
        Fin pour
        Retourner S
    Fin


Fonction Rec(a : entier, b : entier) : entier
    Début
        Si b = 0 alors
            Retourner 1
        Sinon
            Retourner a*Rec(a,    (b-1))
        Fin Si
    Fin
```

Act 2 : Soit une chaine de charon; écrire une f qui nous renvoi l'inverse de cette chaine

```
Fonction Rev(Str : chaine) : chaine
    Début   tmp ← Str 1
        Pour i de 0 a n div 2 faire
            tmp ← Str[i]
            Str[i] ← Str[n-i-1]
            Str[n-i-1] ← tmp
        Fin pour
```

Retourner Soln

Fin

Fonction Rec (ch : chaine, n = 0) : entier) : chaine
    Debut
        Si ch = " " alors
            retourner ""

        Sinon
            retourner (ch[0] + Rec (Sous chaine(ch, 1, Long (ch))

        Fin Si

    Fin

polyjindrone Algo:

Iterative:

Fonction est polyjindrone (ch : chaine) : boolean
    Debut
        i ← 0
        check ← vrai
        tant que i < long (ch) div 2 et check faire
            check ← ch [i] = ch [long (ch)-1-i-1]

            i ← i + 1
        Fin tant que
        retourner check

    Fin

ou :

Fonction polyjin (ch : chaine) boolean
    Debut
        pour i de 0 a Long (ch) - 1 div 2 faire
            Si ch [i] ≠ ch [Long (ch) - 1 - i] alors
                retourner faux

Fin Si

Fin pour

retourner vrai

Fin

Recursive

```
Fonction recString (myString: chaine, l: entier, b: entier, e: entier): booleen
    Debut
        Si b = e ou b+1 = e alors
            retourner Vrai

        Sinon
            Si myString [b] = myString [e] alors
                retourner vrai et recString (myString, l, b+1, e-1)

            Sinon
                retourner faux

            Fin Si
        Fin Si
    Fin
```

```
Fonction prod (p: entier, q: entier): entier
    Debut
        S ← 0
        pour i de 1 à q faire
            S ← S + P

        Fin pour
```

retourner S
Fin

Rec :

Fonction ProdRec (P : entier, q : entier) : entier
  Debut
        Si q = 0 alors
            retourner 0
        Sinon
            retourner   + ProdReg (P , q-1)

        Fin Si

    Fin

Problème 2 :

  Iterative :

    Fonction Sum (m : entier) : entier
        Debut
            S ← 0
            pour i de 1 à m faire
                S ← S + i

            Fin pour
            retourner S

        Fin

Fonction sum (m:entier) : entier
    Debut
        Si m = 0 alors
            retournes 0

        Sinon
            retourner m+ Sum(m-1)

        Fin Si

        Fin

problm 3

Methode Eculide :

Fonction pgcd (a:entier, b:entier): entier
    Debut
        tant que b ≠ 0 faire
            n ← a mod b
            a ← b
            b ← n
        Fin tant que
        retourner a
        Fin

_____

Recursive

Fonction pgcd (a:entier, b:entier):entier
    Debut

- Si b = 0 alors
    retourner a

    Sinon
        retourner (b, a mod b)

    Fin Si

Fin

Méthode de diff

Itérative :

Fonction pgcd (a : entier, b : entier) : entier
    Début
        tant que a ≠ b
            Si a > b alors
                a ⟵ a - b
            Sinon
                b ⟵ b - a
            Fin Si
        Fin tant que
        retourner a
    Fin

Recursive :

Fonction pgcd (a: entier, b: entier) : entier
    Debut
        Si a = b alors
            retourner a
        Sinon
            si a > b alors
                retourner pgcd (a - b, b)
            Sinon
                retourner pgcd (b, b - a)
            ← i
        Fin si
    Fin

---

problème 4:

Itérative:

Fonction isprime (m: entier) : bool
    Debut
        toReturn ← fausse
        i ← 0
        Repeter
            toReturn ← m mod i = 0
            i ← i + 1
        Jusqu'a i = m div 2 out Return

retourner mon tabelium

Fin

Recursive:

Fonction is prime (m: entier, i: entier): bool
    Debut
        { initial value of i is 1, }

        Si m div 2 = i alors
                retourner Vrai

            Sinon
                retourner Vrai et m mod i ≠ 0 et is prime (m, i+1)

            Fin Si

        Fin

Problemes:

Iterative:
procedure Rev (myString: chaine);
    Debut
        pour i de 0 a (Long(myString) - 1) div 2 faire
            temp = myString [i]
            myString [i] ← myString [Long(myString) - 1 - i]
            myString [Long(myString) - 1 - i] ← temp
        Fin pour

        Fin

Recursive
procedure Rev(myString : chaine):
    Debut
        Si Long(myString) > 1 alors
            temp ← myString[i]
            myString[i] ← myString[Long(myString) - 1]
            myString[Long(myString) - 1] ← temp
            Rev(sous-chaine(myString, 1, Long(myString)-2))
        Fin Si
    Fin

problemeb:

Iterative

Fonction pal(ch : chaine): bool
    Debut
        i ← 0
        check ← faune

        Repeter
            check ← ch[i] = ch[Long(ch)-1-i]
            i ← i + 1
        jusqua (Long(ch)-1) div 2 < i ou check

        retourner check
    Fin

Recursive :

Fonction pol (ch : chaine) : bool

    Debut

        Si Len (ch) $\leq$ 1 alors

            retourner Vrai

        Sinon

           retourner Vrai et ch[0] = ch[Long(ch)-1]

                    et pol (sous chaine (ch, 1, long(ch)-2))

        Fin Si

    Fin

probleme 7 :

Iterative :

Procedure Rev (@ t : tab) :

    Debut

    pour i de 0 a Long (t)-1 div 2 faire

        temp $\leftarrow$ t[i]

        t[i] $\leftarrow$ t[Long(r)-1 -i]

        t [Long (t)-1-i] $\leftarrow$ temp

    Fin pour

    Fin

Récursive:

procedure Rev(@t:tab):
   Début
      Si Long (t) > 1 alors
         temp ← t[0]
         t[0] ← t[Long (t)-1]
         t[Long (t)-1] ← temp
         Rev(t[1, Long (t)-2])
      Fin Si
   Fin

Itérative

procedure Rev(@t:tab):
   Début
      P ← 0
      n ← Long (t)-1

      Tant que Long div 2 ≠ P faire
        temp ← t[P]
        t[P] ← t[n]
        t[n] ← temp
        P ← P+1
        n ← n-1

Fin fonction graphe

Fin

Recursive

procedure Rev (@t : tab, p : entier, m : entier)

Debut

   Si longr(t) div 2 < p alors

     temp ← t[p]

     t[p] ← t[m]

     t[m] ← temp

     Rev (t, p+1, m-1)

   Fin Si

  Fin

Problème 8

Rec :

Fonction rev (ch : chaine; m : entier) : chaine

  Debut

   Si m ≥ 0 alors

    retourner

     ch[m] + Rev (ch, m-1)

   Fin si

  Fin

Fin fonction

Si cheats

retourner mid

Sinon

retourner -1

$F_i \ni S_i$

F_i

P

Iterative :

Fonction Rev (ch : chaine) : chaine
Début
    ch2 ← ""
    pour i de long(ch) - 1 allant à 0 fois
        ch2 ← ch2 + ch[i]

    Fin pour

    retourne ch2

Fin

Problème 9 :
Itérative :

Fonction binary (t : tab, b : entier, e : entier, ele)
    Début
        check ← faux

        tant que check = faux et b ≤ e fait
            mid ← (b + e) div 2
            si t[mid]) = ele
                check ← vrai

            Sinon
                si ele > t[mid]
                    b ← b + 1

                Sinon
                    e ← mid - 1

                Fin Si

Recursive :

Fonction binary (t : tab, ele : entier) : entier
    Debut
        Si t [long(t) div 2] = ele alors
            retourne long (t) div 2

        Sinon si t [long (t) div 2] > ele alors
            retourner binary (t [1, (long (t) div 2) - 1], ele)

        Sinon
            retourner long (t) div 2 + binary (t [long (t) div 2) +1,
                                  long (t) - 1]

        Fin Si

    Fin

linéaire :

  Recursive :

  Fonction lineaire (t : tab, n : entier, x, i : entier) : entier
      Si n = i alors
        retourner -1
      Sinon
        Si t [i] = x alors
            retourner i
        Sinon

retourner (t, m, p, i+1)

Fin Si

Fin

Rec 2 : with counter :

Fonction Rec (t : liste, i, p, m) : entier

Début

Si i ≥ m alors

retourner 0

sinon

Si t {i} = p alors

retourner 1 + Rec (t, i+1, p, m)

Sinon

retourner Rec (t, i+1, p, m)

Fin Si

Fin Si

Fin

Fonction binaire (t : tab, m, p : entiers)

début

   d ← 0

   f ← m-1

   P ← -1

   Tant que $d \leq f$ et $p == -1$ faire

      m ← (d+f) div 2

         s ← T{m} == p alors

            P ← m

        sinon

          si $p < T\{i\}$ alors

            f ← m - 1

          Sinon

            d ← m+1

          fin si

          fin tant que

        retourner m

        Fin

Récursive:

Fonction Rec (t : tab, d, f, b : entiers) : entiers

Début

  Si d > f alors

  retournes -1

  Sinon

  m ← (d + f) div 2

  Si t [m] ≥ b
  alors retournes m

  Sinon alors Si t [m] > b alors

  retournes Rec (t, d, m-1, b)

  Sinon

  retournes Rec (t, d, m+1, b)

  Fin Si

  Fin