

Récurseur

Traitement répétitif sans utiliser une boucle -

Condition d'arrêt obligatoire -

ex: 5!

$$\begin{aligned}
 &= 5 \times 4! \\
 &= 5 \times 4 \times 3! \\
 &= 5 \times 4 \times 3 \times 2! \\
 &= 5 \times 4 \times 3 \times 2 \times 1! \\
 &= 5 \times 4 \times 3 \times 2 \times 1 \times \frac{0!}{-1}
 \end{aligned}$$

itératif: utilisation d'une boucle de répétition

factoriel - itératif (n: entier) : entier

Début

$$x \leftarrow 1$$

pour i allant de n à 1 par pas de (-1) faire

$$x \leftarrow x \times i$$

fin pour

retourner x

Fin

x, entier calculer n!
i, compteur, initialisé

ici on décrémente - Il peut s'écrire également en incrémentant

Récursif:

factoriel - récursif (n: entier naturel) : entier naturel

donnée: n, entier naturel, entier pour lequel le factoriel sera calculé -

V.L:

Résultat: n!

Début

si n = 0 alors retourner 1 -

ou si n = 1 alors retourner 1 -

ou si n = 2 alors retourner 2 -

retourner n × factoriel - récursif (n-1)

// condition d'arrêt

Fin



Pile d'appel

1 et	X
	1 × factoriel(0)
	2 × factoriel(1)
	3 × factoriel(2)
	4 × factoriel(3)
	5 × factoriel(4)

jeu
empilement
ordre chronologique



* de la pile
d'appel -

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

Résumé
dépilement



complexité

complexité itératif = $f(n)$ $O(n)$
↳ l'opération en mémoire.

boucle pour
↳ initialisation du compteur : 1 opération.
↳ condition > 1 : test $n+1$, car la condition fausse
↳ $i--$: $2 \times n$

$$\text{Complexité} = 5n + 2 = 3n + 2 + 2n.$$

Complexité récursif - $O(n)$

$$\text{Complexité} = 3n + 2$$

Donner la complexité.

$O(n)$ traitement linéaire, boucle

$O(n^2)$ boucle dans une boucle

$O(\log n)$ réduction par 2 du travail (ex: dichotomie).

~~TD no T~~

1) Esponenziacione ricorsiva

Exponentiel (a; entier, n: entier naturel): entier
données: a, entier n, entier, exposant
VL: i, entier, complexe
p, entier, puissance
résultat: a^m

```

Début p ← 1
| pour i allant de 1 à n inclus faire
| | p ← p * a
| | fin pour
| retourner p

```

) n for's
it
 $\Theta(n)$

complexité décroissante

Expo-réelatif (a: entier, n: entier naturel): entier
derniers: a: entier, n: entier, exprimant
V6:
résultat: a ~ n

Début : si $n=1$ retourner a
retourner $a \times$ expo-rectif ($a, n-1$)

$$\begin{aligned}
 & \underbrace{ax ax a \dots \dots \dots x a}_{n \text{ factors}} \\
 & ax a^{(m-1)} \\
 & ax a^{(m-2)} \\
 & \dots \\
 & ax a^1
 \end{aligned}$$

$$\begin{array}{ccc}
 2 & \leftarrow & 2 \\
 & \downarrow & X \\
 2 \times \exp(-2 \times 1) \\
 2 \times \exp(-2 \times 2) \\
 2 \times \exp(-2 \times 3) \\
 2 \times \exp(-2, h)
 \end{array}$$

Pôle d'appel

Examination TD -

$$\sum_{i=1}^n i^2 = 1 \times 1 + 2 \times 2 + 3 \times 3 + \dots + n \times n$$

somme-itero (m; en t'er): entier
donnée : m; entier

donnée : m : entier
VL : i , entier compris entre 1 et m .
résultat somme i^2 .
si entier naturel, somme

Début - $n \leftarrow 1$
pour i allant de 1 à n inclus faire
 $s \leftarrow s + i$
fin pour

somme récursif (n entier): entier
 Début si ($n=1$) alors retourner 1 // C. A.
 [si $n > 1$ alors facultati] -
 retourner $n \times$ somme récursif ($n-1$)

2) Nombre mystère à 2 joueurs

Consigne

1] prévoire un nbr entre 1 et 100
compris. On nbr mystère
↳ si oui, retourner n° joueur

2] choisir un nbr mystère compris entre 1 et 100

3] Appel licencié -

nb mystère (x, j ; entier) : entier
donné. j entier. $n^{\#}$
demande malicieuse, entier nbr mystère à trouver

V.L. ; y ; entier possible
Réponse: retourner le n° du joueur qui a gagné.

Début

{ si ($x \neq 0$) alors

Il un nbr mystère existe (déjà mis)

faire

afficher 'Saisir une proposition entre 1 et 100'

$y \leftarrow$ saisi

tant que ($y < 1$ ou $y > 100$)

si ($y = x$) alors

) condition d'arrêt

fin si

faire

afficher 'Saisir un nbr mystère compris entre

1 et 100'

$x \leftarrow$ saisi

tant que ($x < 1$ ou $x > 100$)

3) { retourner nb-mystère ($x, (j+1)$ modulo 2)}

Fin

Fibre - recursive terminal (f_a, f_b, m_i en bref) : en bref
 donnée : f_a (s_{n-1})
 f_b (s_{n-2})
 m_i (rang)

Debut

if $n=0$ along returning fib
returner fib - recursive terminal ($f_0 = b$, f_a , $n-1$)

Fin

Exo Concevoir un algo écrivant qui affiche l'ensemble de termes de la suite de Syracuse jusqu'au premier 1 généré.

$$S_m = \begin{cases} \frac{S_{m-1}}{2} & \text{si } S_{m-1} \text{ pair} \\ 3S_{m-1} + 1 & \text{sinon} \end{cases}$$

Espace rc (s. entier) : tableau d'entiers 1D.
donné s, entier, premier terme de la suite (so)
Vé : tab = tab, tableau d'entiers à 1D.

Dibut afficher s

Si ($s=1$) alors ~~lancer la fonction~~ l'écouter ~~la conclusion d'arrêt~~ tab ← allouer mimoire

Si s est pair \leftarrow alors symétrie-rc ($S_1/2$)_{n+1})

sinon tab⁻ Syracuse - rec $(3 + S+1)^{n+1}$

fin si

$b_n \leftarrow s$

utourneer ta b

Ex + TD -

Gx0

Recherche d'heures de travail à Dijon

recherche dans (ou non) une table
recherche dichotomique (tab. tableau 1D, deb; fin v. entiers). cahier
d'activités tab 1 (tab 1) 1. Planification et établissement

données \rightarrow $\{x_i\}_{i=1}^n$ (vecteur) tableau d'entiers
 v , entier, valeur de recherche

VL mil-en en'ter

Debitk milien $\leftarrow \left(\frac{\text{deb} + \text{fin}}{2} \right)$

Si $(\text{tabmif}_{\text{ex}} = v)$ alors retourner milieu

Li (Lösungen) also weiterer wichtiger dichter AD

Sinon retourner rechercher des dictionnaires (tab, tab, million, v)

1D (lab, mlien, fin, v)

Fin mi

Fin

Exo 1

Concevoir un algo récursif qui prend en paramètre d'entrée 2 entiers non nuls et renvoie l'PGCD de ces 2 entiers.

PGCD(a, b : entier naturel) : entier naturel
 donné : a, b, entier naturel, valeur d'entrée. Vu résultat, entier résultat : renvoyer le PGCD de 2 nrs entiers.

Début:

```

  si (a = b) alors
    résultat ← a
  sinon si (a > b) alors
    résultat ← PGCD(a-b, b)
  sinon
    résultat ← PGCD(a, b-a)
  fin si
  retourner résultat
  
```

Fin

Début

```

  si a < b alors a ← b
  si b = c alors retourner a
  retourner PGCD(b, a modulo b)
  
```

Fin

Exo 2

Concevoir un algo récursif Fibonatif (n : entier) : entier qui calcule et renvoie le n^{ème} terme de la suite du Fibonatif

$$F_0 = F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2}, \text{ pour } n \geq 1$$

Fibonatif (n : entier) : entier
 donnée : n, entier, terme du rang

Début si n = 1 ou n = 0

 retourner 1

sinon

 retourner Fibonatif(n-1) + Fibonatif(n-2)

fin si

Fin

Exo 3 Algorithme récursif qui permet d'afficher le contenu d'un tab.

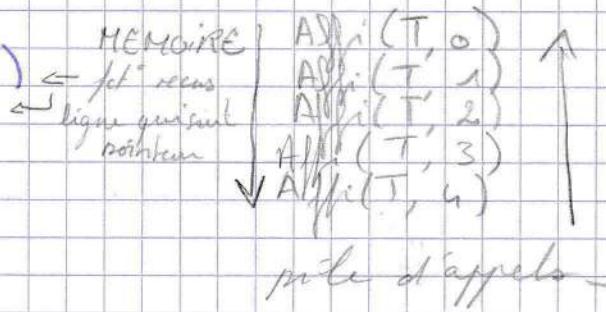
T	5	3	10	35	et
---	---	---	----	----	----

Affichage_rec (T: tableau 1D de taille n)
donnée: T = (Ti) i ∈ [0; n[, tableau à afficher

Début Si (n > 0) alors

pour
de l'ordre Afficher T(n-1)
inverse
fin si

fin



Ex 4 Algo récursif qui retourne la taille d'une chaîne de caractères passée en argument d'entrée.

Chaine b o n j o u r \nul

- Condition d'arrêt arriver à '\n' → si (chaine[i] = '\n')
- Compter un caractère

longueur_chaine_rec (chaine: tableau de caractères): entier
donnée: chaine, chaîne de caractère à analyser

Début si chaine = '\n' alors retourner 0

sinon

retourner 1 + longueur_chaine_rec (chaine+1)

Fin

Ex 4 TD

nbr_chaine (chaine: tableau de caractère): entier
donnée: chaine, chaîne de caractère à analyser

Début si chaine = '\n' alors retourner 0

si chaine est supérieur ou égale à 'A' et inférieur ou égale à 'Z'
retourner 1 + nbr_chaine (chaine+1)

sinon
retourner 0 + nbr_chaine (chaine+1)

Fin

Exo 1

Concevoir un algo récursif qui retourne $\sum_{i=1}^x i + y$
 Le prototype a suivre est:
 calcul(x, y; entier naturel); entier naturel
 donnée: x, y = entiers naturels

Blitz

Réultat: somme d'entiers.

Début si ($x = 0$) retourner y
 Sinon retourner calcul($x - 1, y + x$)
 Fin

Exo 2 Concevoir un algo récursif qui inverse le contenu
 d'un tableau entiers 1D transmis en argument
 d'entrée. Le prototype est:

inverse_tab_1D (T; tableau 1D d'entrée de taille n, i: intia)

données: (T, tableau 1D à modifier (en
 copies) (n, entier, taille tableau)
 i, entier, indice (G \rightarrow D))
 a, entier

V.L

Début si ($i < n/2$) alors
 a $\leftarrow T_i$
 $T_i \leftarrow T_{n-1-i}$
 $T_{n-1-i} \leftarrow a$
 inverse(T, n, i+1)

C
O
R
R
E
C
T
E

Fin

Exo 3 Concevoir un algo récursif permettant de retourner
 le nbr d'occurrences d'un élément "e" dans un
 tableau 1D de taille n, passé en argument d'entrée
 nb_occ_tab_1D (t; tableau 1D, n; entier, e; entier entier
 donnée; t: (t_i) i \in 0..n-1 tableau qd parcouru
 n, entier taille tableau E
 e, entier, l'élément à rechercher -

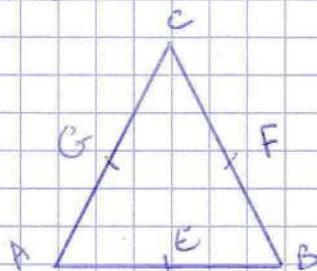
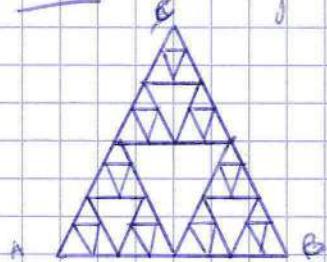
Début

si ($n = 0$) alors retourner 0
 si ($t_{n-1} = e$) alors
 retourner 1 + nb_occ_tab_1D (t, n-1, e)
 Sinon
 retourner nb_occ_tab_1D (t, n-1, e)

fin si

Fin

D7 Ex 9 Triangle de Sierpinski



Sierpinski ($x_a, y_a, x_b, y_b, x_c, y_c, n$: entiers)
données (x_a, y_a) coordonnées du point A

(x_b, y_b) " " " B

(x_c, y_c) " " " C

VL (x_e, y_e), coordonnées du point E
 (x_f, y_f) " " " F
 (x_g, y_g) " " " G

Début

Si ($n=0$) alors calcul coordonnées du milieu.

$$x_e = \left(\frac{x_a + x_b}{2} \right)$$

$$y_e = \left(\frac{y_a + y_b}{2} \right)$$

$$x_f = \left(\frac{x_c + x_b}{2} \right)$$

$$y_f = \left(\frac{y_c + y_b}{2} \right)$$

$$x_g = \left(\frac{x_c + x_a}{2} \right)$$

$$y_g = \left(\frac{y_c + y_a}{2} \right)$$

tracer - triangle ($x_g, y_g, x_e, y_e, x_f, y_f$)

Il afficher 3 lignes entre les points
// autre jet

Sierpinski ($x_a, y_a, x_e, y_e, x_g, y_g, n-1$)

// AEG

Sierpinski ($x_e, y_e, x_b, y_b, x_f, y_f, n-1$)

// EBF

Sierpinski ($x_e, y_e, x_g, y_g, x_f, y_f, n-1$)

// CGF

fin n
Fin

Exo A Suite de Conway

1

1 1

2 1

1 2 1 1

1 1 1 2 2 1

3 1 2 2 1 1

\Rightarrow une fois 1, une fois 2, deux fois 1.
 $\hookrightarrow 1 \quad 1 \quad 1 \quad 2 \quad 1$

Conway (src : tableau 1D, taille : entier, m entier)

demandé src = (src_i) i ∈ [0...n] tableau à afficher
taille, entier, taille de src
m, entier, rang

VL : tab = (tab_i) i ∈ [0...m] tableau de la ligne suivante
m, entier, taille de tab
i, entier, compteur de boucle

Début // Etape 1: afficher le contenu de src

pour i allant de 0 à taille (exclu) faire

Afficher src_i

si (src_{i+1} = src_i) alors m ← m + 1

fin pour

// Etape 2

tab ← allonge minceur (m)

// Etape 3: remplir tab

// Etape 4: appel récursif.

Exo A Suite de Conway

1

1 1

2 1

1 2 1 1

1 1 1 2 2 1

3 1 2 2 1 1

→ une fois 1, une fois 2, deux fois 1.
↳ 1 1 1 2 2 1

Conway (src : tableau 1D, taille : entier, m entier)

donnée src = (src_i) i ∈ {0...n} tableau à afficher
taille, entier, taille de src
m, entier, rang

VL: tab = (tab_i) i ∈ {0...n} tableau de la ligne suivante
m, entier, taille de tab

cpt, entier, compteur de boucle
cpt, entier, compteur de l'élément / c, entier, indice tableau tab

Début // Etape 1: afficher le contenu de src

pour i allant de 0 à taille (exclus) faire

Afficher src_i

* m ← i
si (i = n) alors
retourner
afficher src_i

si (src_{i-1} = src_i) alors m ← m + 1 // calcul la taille
du prochain tableau à afficher

fin pour

// Etape 2
tab ← alloue mémoire (m)

// Etape 3: remplir tab cpt ← 1 | c ← 0
pour i allant de 1 à taille (exclus) faire

si (src_{i-1} = src_i) alors cpt ← cpt + 1

sinon tab_c ← cpt

tab_{c+1} ← src_{i-1}

c ← c + 1

fin sinon/si

fin pour

tab_c ← cpt

tab_{c+1} ← src_{i-1}

) // dernières cases du tableau tab -

Afficher ↴

libérer (src)

conway (tab, m, n-1)

fin si

Exo 1

Concevoir un algo récursif qui retourne $\sum_{i=1}^x i + y$
 Le prototype à suivre est:
 calcul(x, y : entiers naturels) : entier naturel
 donnée : x, y = entiers naturels

Blitz

Réultat : somme d'entiers

Début si ($x = 0$) retourner y
 sinon retourner calcul($x - 1, y + x$)
 Fin

Exo 2 Concevoir un algo récursif qui inverse le contenu d'un tableau 1D transmis en argument d'entrée - Le prototype est :

inverse_tab_1D (T : tableau 1D d'entrée de taille n, i : entier)
 données (T, tableau 1D à modifier (en place),
 copies (n, entier, taille tableau),
 VL i, entier, indice (G → D)
 zéroat a, entier

Début si ($i < n/2$) alors | C
 a $\leftarrow T_i$ | O
 $T_i \leftarrow T_{n-1-i}$ | R
 $T_{n-1-i} \leftarrow a$ | E
 inverse(T, n, i+1) | T
 Fin

Exo 3 Concevoir un algo récursif permettant de retourner le nbr d'occurrences d'un élément "e" dans un tableau 1D de taille n, passé en argument d'entrée
 nb_occ_tab_1D (t : tableau 1D, n : entier, e : entier entier)
 données : t : (t_i) i ∈ 0..n-1 tableau qd'entiers
 n, entier taille tableau
 e, entier, l'élément à rechercher

Début si ($n = 0$) alors retourner 0
 si ($t_{n-1} = e$) alors
 retourner 1 + nb_occ_tab_1D (t, n-1, e)
 Sinon retourner nb_occ_tab_1D (t, n-1, e)
 fin si
 Fin

Ex1 Concevoir un algo récursif qui retourne 1 si un nbr n (passé en entrée) est pair, 0 sinon.

paire (n: entier naturel): entier
donnée: n: entier naturel

Début

```
    si (n > 1)
        si (n = 0)
            retourner 1
        si (n = 1)
            retourner 0
        retourner paire (n - 2)
```

Fin

Ex2 Concevoir un algo récursif permettant de calculer et retourner la somme des nombres positif d'un tableau d'entier 1D de taille n passé en entrée

Nb_positif (t: tableau 1D d'entier, n entier): entier
donnée: t: tableau 1D d'entier
n: entier, taille du tableau

Début si (n = 0) retourner 0

```
    si (n > 0)
        retourner Nb_positif (t, n-1) + tn
    sinon
        retourner Nb_positif (t, n-1)
```

Fin

Ex3 Concevoir un algo récursif terminal qui prend en entrée 2 entiers a et b, et renvoie la somme de tous les a entiers multiples de 3 compris strictement entre a et b

somme_multi3 (a, b res; entier)

données a; entier

b; entier

res; entier, contient la somme de ts les entiers multiples de 3 compris entre a et b.

VL: e, entier, contient a+1

Début e ← a+1

```
    si e = b alors retourner res
    si (e modulo 3 ≠ 0) alors e ← e
        // la valeur n'est pas multiple de 3
    fin si
```

return (somme_multi3 (a+1, b, res+e))

Fin

Début si (a+1 = b) alors retourner res

```
    si (a+1) modulo 3 = 0 alors
        retourner somme_multi3 (a+1, b, res+a+1)
    sinon
        retourner somme_multi3 (a+1, b, res)
```

STRUCTURES

Définir un nouveau type de données.

Syntaxe algo

type structure nom de la structure
 type variable variable

 nom : chaîne de caractères type de la variable
 nom de variable
 prénom : chaîne de caractères
 date de naissance : chaîne de caractères
 fin type

Déclaration

VL: p, personne

Accès aux champs (Variable de la structure)

① pas de retour de variable (var accessible uniquement au sein de l'algo)
 p → nom ← "Foto";
 p → prénom ← "Jean";

Exemple d'Algo

② retour de résultat

saufie - personne (): personne

VL: p, personne

Début p ← réservé mémmoire personne

p → nom ← "Foto";

p → prénom ← "Jean";

p → date de naissance ← "01/01/1990";

retourner p.

Fin

TD n°8

Exo 1

Type structure complexe

re, réel, partie réelle du nombre complexe

im, réel, partie imaginaire du nombre complexe

fin type

① Afficher un nombre complexe

affichage (c: complexe)

donnée: c, complexe, le nombre complexe à afficher.

Début

afficher $c \rightarrow re + ' + im + 'i$

// $c \rightarrow re + ' + im + 'i \Rightarrow re + im + 'i$

// $a + bi$

Fin

② Addition de 2 nombres complexes

addition complexe (c1: complexe, c2: complexe): complexe

données: c1, complexe

c2, complexe

VL: c, complexe, contenu $c1 + c2$

Début

c \leftarrow réservé mémoire complexe

c \rightarrow re $\leftarrow (c1 \rightarrow re) + (c2 \rightarrow re)$

c \rightarrow im $\leftarrow (c1 \rightarrow im) + (c2 \rightarrow im)$

retourner c

Fin

③ Retourner la partie réel d'un nombre complexe

Imaginaire

Partie -im (c: complexe): réel

donnée: c, complexe, nombre dont on retourne la partie réelle

imaginair

Début

retourner c \rightarrow re

retourner c \rightarrow im

Fin

④ Vérifier si deux nombres complexes sont égaux

égalité-complexe (c1, c2: complexe): booléen

données: c1, complexe, nbr complexe à vérifier

c2, complexe, nbr complexe à vérifier

Résultat: vrai si $c1 = c2$, sinon faux

Début si $((c1 \rightarrow re) = (c2 \rightarrow re)) \text{ et } ((c1 \rightarrow im) = (c2 \rightarrow im))$

T alors retourner vrai

Fin sinon retourner faux

⑤ Retourner l'opposé d'un nombre complexe.

opposé-complexe (c : complexe): complexe
 donnée: c , complexe, le nombre complexe
 VL: c_1 , contenu l'opposé de c ($c_1 = -c$)

Début $c_1 \leftarrow$ réservé mémoire complexe.

$$(c_1 \rightarrow \text{re}) \leftarrow -(c \rightarrow \text{re})$$

$$(c_1 \rightarrow \text{im}) \leftarrow -(c \rightarrow \text{im})$$

retourner c_1 .

Fin

Sans variable aléatoire

$$c \rightarrow \text{re} \leftarrow -(c \rightarrow \text{re})$$

$$c \rightarrow \text{im} \leftarrow -(c \rightarrow \text{im})$$

retourner c .

Afficher

⑥ Retourner la soustraction de 2 nbr complexes

soustraction-complexe (c_1 : complexe, c_2 : complexe): complexe
 données: c_1 , nbr complexe
 c_2 , nbr complexe

VL: c , complexe, mémoire ($c_1 - c_2 = c$)

Début $c \leftarrow$ réservé mémoire complexe.

$$(c \rightarrow \text{re}) \leftarrow (c_1 \rightarrow \text{re}) - (c_2 \rightarrow \text{re})$$

$$(c \rightarrow \text{im}) \leftarrow (c_1 \rightarrow \text{im}) - (c_2 \rightarrow \text{im})$$

retourner c .

Fin Afficher $c \rightarrow \text{re} + ' + c \rightarrow \text{im} + 'i'$

ou \hat{m}

Afficher $((c_1 \rightarrow \text{re}) - (c_2 \rightarrow \text{re})) + ' + ((c_1 \rightarrow \text{im}) - (c_2 \rightarrow \text{im})) + 'i'$

⑦ Inverse d'un complexe

inverse-complexe (c : complexe): complexe

donnée: c , complexe

VL: inv , complexe, contient l'inverse de c

Début $inv \leftarrow$ réservé mémoire complexe

$$(inv \leftarrow \text{im}) \leftarrow (-c \rightarrow \text{im}) / ((c \rightarrow \text{re}) \times c \rightarrow \text{re}) + (c \rightarrow \text{im}) \times (c \rightarrow \text{im})$$

$$(inv \leftarrow \text{re}) \leftarrow (c \rightarrow \text{re}) / ((c \rightarrow \text{re}) \times c \rightarrow \text{re}) + (c \rightarrow \text{im}) \times (c \rightarrow \text{im})$$

retourner inv

fin

⑧ Multiplication de 2 nbr complexes

multiplication (c_1 : complexe, c_2 : complexe): complexe

donnée: c_1, c_2 , complexe

VL: $mult$, complexe, contient le résultat de $c_1 \times c_2$

Début $mult \leftarrow$ réservé mémoire complexe

$$(mult \rightarrow \text{re}) \leftarrow (c_1 \rightarrow \text{re}) \times (c_2 \rightarrow \text{re}) - (c_1 \rightarrow \text{im}) \times (c_2 \rightarrow \text{im})$$

$$(mult \rightarrow \text{im}) \leftarrow (c_1 \rightarrow \text{re}) \times (c_2 \rightarrow \text{im}) + (c_2 \rightarrow \text{re}) \times (c_1 \rightarrow \text{im})$$

retourner $mult$

Fin

① Coordonnées polaires :

② Type structure polaire

mod : réel, module du nbr complexe

arg : réel, argument du nbr complexe

③ Concevoir un algo pour convertir un nbr complexe sous sa forme polaire

Conv. complexe polaire (c : complexe); polaire donnée; c , complexe

VL: p , polaire, contient la forme polaire de c

Début $p \leftarrow$ réserv. mémoire polaire

| $(p \rightarrow \text{mod}) \leftarrow$ module (c)
 $(p \rightarrow \text{arg}) \leftarrow$ arctan $\left(\frac{c \rightarrow \text{im}}{c \rightarrow \text{re}} \right)$

Fin

④ Concevoir un algo pour convertir un nbr complexe polaire vers sa forme

conv. polaire - cart (p : polaire) complexe

donnée: p , polaire

VL: c , complexe, contient la forme cartésienne de p .

Début $c \leftarrow$ réserv. mémoire complexe

| $(c \rightarrow \text{re}) \leftarrow (p \rightarrow \text{mod}) \times \cos(p \rightarrow \text{arg})$
 $(c \rightarrow \text{im}) \leftarrow (p \rightarrow \text{mod}) \times \sin(p \rightarrow \text{arg})$

Fin

⑤ La somme des racines d'unité

$$2\pi i k/n = \cos\left(\frac{2k\pi}{n}\right) + i \sin\left(\frac{2k\pi}{n}\right)$$

$k=0$

$$\sum_{k=0}^{n-1} e^{2ik\pi/n}$$

Racine unité (n: entier): complexe

donnée: n , entier négatif

VL: somme, complexe, contient la somme des racines

k , entier, compris en

Début

| somme \leftarrow créer-complexe (0, 0)

| pour k allant de 0 à $(n-1)$ inclus

Tourne

| $c \leftarrow$ créer complexe $\cos\left(\frac{2k\pi}{n}\right), \sin\left(\frac{2k\pi}{n}\right)$

| somme \leftarrow ~~addition~~ addition complexe (somme, c)

fin pour

retourner somme

Fin

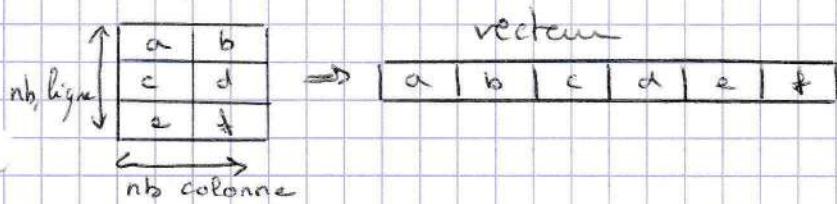
creer-complexe (a: réel, b: réel) : complexe
 donnée, a, réel, partie réelle
 b, réel, partie imaginaire
 VL: c, complexe

Début

c ← réservé complexe
 $(c \rightarrow \text{re}) \leftarrow a$
 $(c \rightarrow \text{im}) \leftarrow b$
 retourner c

Fin

1.2 Représentation d'un tableau 2D (matrice)



type structure

nb lignes : entier, nbr de ligne
 nb colonne : entier, nbr de colonne
 tab : tableau d'entier à 1D.

① Algo qui crée un élément de type matrice

creer-matrice (n: entier m: entier) : matrice
 données : n, m : entier, nb de lignes
 m, entier, nb de colonnes
 VL: mat, matrice

Début

mat ← réserve mémoire matrice
 $(\text{mat} \rightarrow \text{nb ligne}) \leftarrow n$
 $(\text{mat} \rightarrow \text{nb colonne}) \leftarrow m$
 $(\text{mat} \rightarrow \text{tab}) \leftarrow \text{réserve mémoire } (n \times m)$
 retourner mat.

Algo qui modifie l'élément (i, j) de la matrice

modif recuper-element (mat: matrice, i, j: entiers) ^{valeur: entier}
 données i : entiers ^à indices de l'élément
 j : entiers ^à récupérer
 mat : matrice ^{modifier valeur, entier, new value of l'élément}
 VL: indice, entier, indice de l'élément (i, j) de la matrice

Début indice ← i * mat → nb colonne + j

retourner ($\text{mat} \rightarrow \text{tab}[\text{indice}]$)

écrase indice dans le vecteur tab ($\text{mat} \rightarrow \text{tab}[\text{indice}] \leftarrow \text{valeur}$)

Exo 1 Concevoir un algo permettant de calculer et retourner Bliby -
une nbr complexe

type structure complexe
 re : réel
 im : réel
fin type

module (c: complexe) réel
 donnée : c: complexe, nbr complexe

Début retourner $\sqrt{(c \rightarrow u)(c \rightarrow re) + (c \rightarrow im)(c \rightarrow im)}$

Fin

Exo 2 1/ Définir la structure point permettant de stocker
 les coordonnées d'un point
2/ Concevoir un algo qui affiche les coordonnées d'un
 point passé en entrée

type structure point
 x: réel
 y: réel
fin type

Affichage - point (p: point):

 Donnée: p; point

Début Afficher ('+' + p \rightarrow x + ', ' + p \rightarrow y + ')'

Fin

Exo 3 Soit la structure "heure" qui décrit une heure donnée
à travers 3 champs (heure, minute et seconde)

type structure heure
 hh: entier
 mm: entier
 ss: entier
fin type

heure_seconde (a: heure): entier
 donnée a; , heure -

Début

 retourner (a \rightarrow hh) \times 3600 + (a \rightarrow mm) \times 60 + (a \rightarrow ss)

Fin

Exo1 Concevoir un algo qui saisie et retourne une heure

Type structure heure

hh : entier

mm : entier

ss : entier

fin type

Saisie () : heure

VL: h, heure

Début h \leftarrow réservé mémoire heure

h \rightarrow hh \leftarrow saisie

h \rightarrow mm \leftarrow saisie

h \rightarrow ss \leftarrow saisie

retourner h

Fin

Blitz

Exo2 Concevoir un algo qui comp e 2 heures et retourner la plus grande de d.

Type structure heure

hh : entier

mm : entier

ss : entier

fin type

comparer (a:heure, b:heure) : heure

donnée: a: heure

b: heure

Début Si $((a \rightarrow hh) > (b \rightarrow hh))$

retourner a

Si non si $((a \rightarrow mm) > (b \rightarrow mm))$

retourner a

Si non si $((a \rightarrow ss) > (b \rightarrow ss))$

retourner a

Fin si Fin si non

retourner b

Fin

Ex1

Blitz

- 1/ Définir la structure "vecteur" permettant de décrire un tableau ayant n valeurs ex $V(1, 2 \times 5, 3, 4 \times 5)$ est un vecteur de 4 éléments réels
- 2/ Concevoir un algo qui permet d'allouer l'espace mémoire d'un vecteur en fonction du nbr d'éléments

type structure vecteur

tab : tableau d'entier

n : taille de tab.

fin type

alloare - vecteur (n: entier) vecteur

données : n : entier, taille vecteur

V : V : vecteur

Début V ← réserve mémoire vecteur

(V → n) ← n

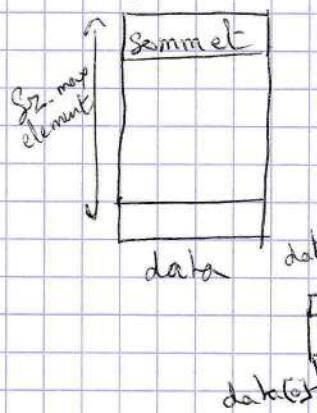
(V → tab) ← réserve mémoire tableau de n éléments

renvoyer V

Fin

Ex2 Concevoir un algo qui initialise l'ensemble des éléments d'un vecteur avec un tableau de réel passé en argument d'entrée

Pile



Pile :

- limitée à sz_max_element
- éléments d'un tab
- indice du sommet

} type structure pile

- pile vide aucun élément
 $\hookrightarrow \text{sommet} = -1$

Type structure pile

```

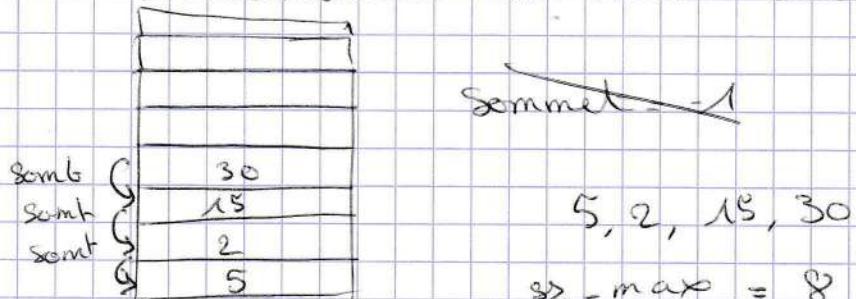
| sz_max : entier, nbr max d'élément de la pile
| sommet : entier, indice
| data : tab de sz_max éléments
fin type
    
```

• Empiler un élément

ssi sommet < sz_max
 \hookrightarrow incrémenter la valeur du sommet puis ajout de l'élément

• Dépiler un élément

ssi sommet > 0 ~~retirer l'élément~~
 \hookrightarrow décrementer la valeur du sommet



Début

```

p ← réservé mémoire pile
(p → sz_max) ← n
(p → sommet) ← -1 // pile
(p → data) ← réservé mémoire (n : entier)
    
```

retourner p

fin

- 1) Concevoir un algo qui vérifie si une pile est vide
retourner vrai sinon faux
est-pile-vide (p: pile) : boolean
- 2) Algo qui vérifie si une pile est pleine retourne vrai
sinon faux
est-pile-pleine (p: pile) : boolean
- 3) Algo qui ajoute un élément e à la pile et retourne vrai si l'ajout se fait, sinon faux
empiler (p: pile, e: entier) : boolean
- 4) Algo qui retire un élément de la pile, Retourne vrai si le dépilement se fait, sinon faux
Dpiler (p: pile, e: entier) : boolean

- 1) est-pile-vide (p: pile) : boolean
donnée : p, pile, pile à vérifier

Déb
 si ($p \rightarrow \text{sommet} = -1$) alors
 retourner vrai
 sinon retourner faux
Fin

- 2) est-pile-pleine (p: pile) : boolean
donnée : p, pile

Déb
 si ($p \rightarrow \text{sommet} = (p \rightarrow sz_max - 1)$)
 alors retourner vrai
 [sinon]
 retourner faux
Fin

- 3) empiler (p: pile, e: entier) : boolean
donnée modifiée : p, pile, pile à traiter
donnée : e, entier, élément à empiler

Déb si est-pile-pleine (p) alors
 retourner faux
 sinon
 $(p \rightarrow \text{sommet}) \leftarrow (p \rightarrow \text{sommet} + 1)$
 $(p \rightarrow \text{data}(p \rightarrow \text{sommet})) \leftarrow e$ // en c $p \rightarrow \text{data}[p \rightarrow \text{sommt}]$
 retourner vrai
 fin si.
Fin

a) Dépiler (p: pile, e: entier): booléen
données modif: p, pile
e, entier

Déb si (est-pile-vide (p)) alors retourner faux
sinon
 e \leftarrow (p \rightarrow data)
 (p \rightarrow sommet) \leftarrow (p \rightarrow sommet - 1)
 retourner vrai
Fin

File



- limitée à sz_max éléments
- in, entier indice d'entrée
- out, entier indice de sortie
- data, tab de sz_max éléments
- sz, entier, la taille courante de la file

• file vide

$$\text{ssi } sz = -1 \\ \text{in} = \text{out} = -1$$

* Enfiler ssi $sz < sz_{\text{max}}$

 |
 |
 | incrementer sz
 | in \leftarrow sz

* Dépiler

- Retirer l'élément à l'indice out
- Décaler les éléments vers la droite d'une position
- Actualiser sz et in // Décrementation

			8	7	5	3	$e = 3$
in	sz					out	

→ 4) Algo qui ajoute un élément e de la file enfiler (f: file, e: entier): booléen

1) Algo qui crée une nouvelle file
nouvelle_file (n: entier): file

2) Algo qui vérifie si la file est vide
est_file_vide (f: file): booléen

3) Algo qui vérifie si la file est pleine
est_file_pleine (f: file): booléen

5) Algo qui retire un élément de la file
dépiler (f: file, e: entier): booléen

1) Nouvelle - file (n: entier) : file
donnée : n, entier, nombrelement de la file
Vé : f, file

Deb
| $f \leftarrow$ réservé mémoire file
 $f \rightarrow sz \leftarrow max$ ← n
 $f \rightarrow data \leftarrow$ réservé mem (n: entier)
 $(f \rightarrow sz) \leftarrow (f \rightarrow in) \leftarrow 1$
retourner f $(f \rightarrow out) \leftarrow 0$
Fin

2) est - file - vide (f: file) : boolean
donnée f: file à traiter

Deb
si $(f \rightarrow sz = 0)$ et $(f \rightarrow in = f \rightarrow out)$
alors retourner vrai
sinon
retourner faux
Fin

ifajouter (f: file, e: entier) boolean
do modif f, file
do ! f e, entier, élément à ajouter

Deb
si est - file - pleine (f) alors
retourner faux
sinon
 $(f \rightarrow sz) \leftarrow (f \rightarrow sz) + 1$
 $f \rightarrow in \leftarrow (f \rightarrow sz)$
 $f \rightarrow data \leftarrow e$
retourner vrai
Fin

3) est - file - pleine (f: file) boolean
do file à traiter

Deb
si $(f \rightarrow sz) = ((f \rightarrow sz - max) - 1)$ alors
retourner vrai
sinon
retourner faux
Fin

4) Defiler (f: file, ei entier) : boolean