Sur internet https://goo.gl/Hzol-MBK INFORMATIQUE

- Programmer
- Machine, programmeur, l'algorithme, programme.

Homme → Machine (Def° outil)
pilote ↳ amplifie
↳ + rapide (ex type calcule)

piloter mais pas programmable → commander

Piloter - sequence d'instruction pdt le tps qui coule.
- notre choix -

Programmer - (ex avion - pilote automatique)
- enregistrement d'instruction d'une machine.
séquence determinee sont comportement sa trajectoire.

★ Pascaline 1645
> 1er calculatrice
⇒ Non programmable, mais opérant sur commande.

★ Metier Jaquard 1801
> métier à tisser
⇒ automatisation, crochets guidés (1er esquisse de programme)
⇒ programme : carte perforé.

Revolut° Industriel l'H a peur que la machine vole du travail. ⇒ Revolte des Canuts.

★ La machine differentielle de Babbage (codage) 1830
> Machine differentielle
> New Pascaline.
⇒ Table de calcul sans erreurs.

★ La machine analytique Babbage 1834
> 1er ordinateur
⇒ Intègre jaquard, machine diff, rendre programmable

ARCHITECTURE BABBAGE
- Moulin → processeur (calcule)
- Magasin → memoire, disque dur (stock data)
- Impression → ecran (donne les resultats)

Carte perforer ⇒ coder directement en Binaire.

★ Bombe electromécanique de Turing 2nd GM.
> Machine cryptage
Decoder les codes all changer tlis jrs.

Def° mt mathematique
Automate à état
decrit comportemt Machine

ERE MODERNE
democratisat° des machines.
ADA LoveGrace 1er programmeuse

• Not° Algo
- Stratégie, façon de s'y prendre pr résoudre 1 pb.
- analyser en entré
- résoudre en sortie.

* Equifinalité
résoudre plusieurs stratégie pr atteindre 1 but.
stratégie la + performante ! Enjeux fondamental
Ex: marché de profit.
Ê en mesure face a des pb d'identifier laquelle est la + performante. En tant que programmeur.
Meilleur en tant que performance.
- optimisat° de memoire d'energie
- optimisat° du tps. + vite possible

• Language algorithme => pseudo code.
• Language interprétable par la machine => micro processeur.

Algo → Language de programmat° tech accessible à l'H. => Outil Compilateur (Binaire) (.exe) => language machine (lu en binaire)

Ingénieur
- Curieux - savoir comment ça fct° - analiste pb.
Ap avoir compris le pb. Imaginer solut°. Stratégie.
Capacité de concrétiser / réaliser la solution.

ANALYSE => CONCEPTION

réaliser => implementer.

* Pointeur * Programmation recursive (ex factoriel)
* Algo arbre 4 projets ds l'année. - info & calcul (bac) - plateau de 2 - plateau jeux strategie - approfondissemt Algo (fractale)

2 semestre 4 trimestre.

3 TP + 1 piscine + lancemt projet

Exam: Théo
1 controle intermédiaire + 1 soutenance projet
1 partiel (janv)
Prat
2 exams indiv
2 soutenances projets

ALGORITHME COURS - TD -

main - fct principale
- hyrarchie de fct

modulariser son prog. : morceau autonome ⇒ qu'on imbrique ∈

RpL : nbr premier divisible par 1 et par lui-même

Crible : prog qui recherche les nbrs premier

- Algo prin
- problème
- sous problème

Teste de primalité ds une fct spécialisée

fct principale appel fct° sec
fct principale est suspendue à la durée d'execut° de la fct seconde
fct sec return fct principale variable locale

---

sémantique : se que fait le proj

Syntaxe : > nom
> liste d'argument / paramètre

Algo (x: entier)
ou bien Algo (x: entier, y: entier)

fct sans argument ⇒ procédure Algo()

en C

```
Procedure ();
si est pair (5+x) alors ...
z ← somme (x, y);
retourner racine (carre (x) + carre (ze));
si on (et (non (x=y), y=z), et (x=y, non (y=z))) alors...
```

$f(x) = xe^2$
$g(x) = \sqrt{xe^2}$
gof

ou retourner racine (carre (x) + carre (ze)) // gof
si on (et (non (x=y), y=z), et (x=y, non (y=z)))) ...

---

fct princ main
→ déléguer les tâches à des fct auxiliaires
passer des données à cette fct
→ TABLEAUX

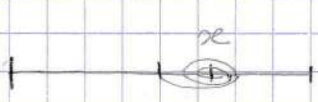
1) $n! = [1 \times 2 \times \dots \times (n-1) \times n]$
2) $n! \begin{cases} 1 \text{ si } n = 0 \\ n \times (n \times -1) \end{cases}$

1) Factorielle (n: entier): entier
Donnée : n, l'entier dont on retourne la fact
Variable locale : res, un entier le résultat
i, un compteur
Début
| res ← 1
| Pour i allant de 1 à n faire res ← res × i
| retourner res
Fin

même prog en résultat récursif ⇒ SS Boucle

2) Factorielle (n. entier): entier
Donnée : idem
Début
| si n = 0 alors retourner 1
| [sinon] retourner n Factorielle (n-1)
Fin

Dicotomie Racine (x : réel > 1)



Donnée : x : réel

Var locale : y réel approximat° actuelle de la racine
min, max, bornes inf et sup de l'intervalle de recherche.

```
Début ; min ← 0
        max ← x
        faire tant que y ← (min + max) / 2
          si y×y > x alors max ← y
          sinon si y×y < x alors min ← y
          sinon retourner y
Fin
```

```
float racine (float x, float precision) {
   float min = 0 ;
   float max = x ;
 do {
      y = (min + max) / 2 ;
      if (y * y < x) min = y ;
       else max = y ; }
 while (min != max) ;          /        while (min - max < precisi°)
 return y ; }
```

```
print (" racine (2) = %f\n", racine (2,0, 0,0001)) ;
```

Suite Condition / Operateur Connecteur logique — ALGO

Algo	Lang C
>	>
<	<
≥	>=
≤	<=
=	==
≠	!=

CONNECTEUR LOGIQUE

	Prog C
OU	||
ET	&&
NON	!

Repetition avec Boucle ttq et ftq

Repetit° d'un affichage 100 fois.

Solut° > 1 instruction pour...
- ... demander de repeter 100 fois l'instruct°
> 3ème structure de contrôle.

Instruct°
* Eval VRAI ou FAUX
* Si Vrai
  1 - execution instruct° contrôlée
  2 - reprend son execut° à partir de 1.

Compteur de Boucle.
* Incremente à chaque passage
  - On parle
  - Un algo qui utilise une boucle algo iteratif

Tant que condit° faire instruct° fin.

C// `while (condition) instruction;`

```
while (condition) {
    instru 1;
    instru 2; }
```

```
int main () {
  int i=0;
  while (i<100);
    i = i+1;
  ou i = i++
Debut print...
  i ← 0
tant que i<100 faire
  i ← i+1
  afficher "Hello"
```

i ← i+1 incrémenter
i ← i-1 decrementer

NBRE MYSTERE () Donnée : x entier
Variable locales n : entier
Résultat : dessiner un nbr mystère
Debut afficher "saisir un nbr"
n ← saisir
tant que n ≠ x faire
afficher "Perdu. Saisir un autre nbr"
n ← saisir
fin
Afficher gagné
Fin

```
#include <studio.h>
void nbr_mystere (int x)
{ int y;
  printf("saisir un nbr:\n");
  scanf("%d", &y)
  while (x != y) {
    printf("Perdu \n");
    scanf("%d", &y);
  printf("Gagné \n");
```

i = i + 1
i++; → incrementation de 1 en 1
i += k; → i += k incrementation de k en k.

i--
multiplier x / * i *= 2; i /= 2;

Obtenir le reste de la division euclidien 17 % 5
modulo

Faire tant que
while
de instruction (condition);
do { Instruct° 1;
Instruct° n; }
While (condition);

ENTRÉE / SORTIE

Syntaxe: # define (directive d'un pré compilateur)
MAX_SIZE → une chose que 1024

Voir type de données

- int getchar() ou getc(stdin)
récup un caractère après l'autre - (ex: 6 getchar pr 6 caract)

Passer un caractere ex: a minuscule en A majuscule

Utilise la fct° Upper (be : symbole); symbole
Donnée ce le caractère capitalisé
début
retourner (be - 'a') + 'A'
fin

Retourner Instruct° fondamentale
appel de variable

2 fonctions : - fini interrompre une progr (si dem : 0 alors return 0)
: retourner une valeur
Conversion caractere chaine %c

Astuce - espace devt %c
Saisie securisé while (rep != 'o' && rep != 'n');

TABLE ASCII

```
void ascii_table(){
    int i = 0;
    while (i < 256){
        printf("%d) %c", i, i);
        i++; }
}
```

Conjecture de Teller

- Processeur : exe instruction
- Mémoire : stockage & réutilisat° data
- Interface : interaction H-f

• Programme
> squence d'instruction exé par l'utilisateur
- avg Debut - Fin
> Instruction contrôle interface
- instruct° Entrée - Sortie
> Instruct° controle mémoire
- instruct° d'affectat° et évalt de var.

Structure de l'Algo

1- Entête [declarat° algo] (nom, E/S, var, résultat qu'il produit)
2- Le corps [sequence d'instruct° qui le définit]
- Début
- fin

nom ← saisir (affectat°)
transferer "saisir" ds nom
+ concatener - complete avec la valeur de var

variable, type primitif

char → caractère
int → entier
float → reel
chaîne de caractère utilisat° tableau

Saisir: scanf ("%s", nom); // saisie du nom par l'utili

commentaire // /**
* Commentaire
*/

achiffre x + ' + ' + y + ' = ' + (x + y)
⇒ Afficher var x + var y = var (x + y)

Char → %c
Entier → %d
Chaîne de caractère → %s

printf ("enregistrement n° %d du %s", num, date);

Connaître la taille d'un type de donnée
printf ("taille d'un int : %d", sizeof(int));

Branchement conditionnel
Expression de condition -

permet de l'algo - evalue la situation, des conditions - oral : variable
permet de choisir entre tel et tel sequence d'instruct°
adaptation du prog en fct° des circonstances.

Algo = stategie de resolution de prob
~~Condition~~ = expression mathematiques (variable operateur num, anal)
Condition / proposition logique VRAI / FAUX (boulet 1)

Algo : nbr mystere (x: entier)
Debut
| afficher 'saisir un nbr'
| y ← saisir
| Si x = y alors afficher 'Gagné'
| Sinon afficher 'Perdu'
Fin

Si obligatoire pr un sinon
Sinon est facultatif.

Le C utilise une copie

#include <stdio.h>
// l'argument x est l'entier à deviner
void nbr_mystere (int x) {
int y;
printf("saisir un nbr : \n");
scanf("%d", &y);
if (x == y) printf("Gagné ! \n");
else printf("Perdu ! \n"); }

Tabulat°
\n : retour à la ligne

&y espernet devant la variable

Si j'utilise un scanf les variable qui s'agit d'affecter, tjrs un espernet
Exception avec des variable de type tableau ou chaîne pas espernet

fct° qui ne renvoie a aucun resultat estune procédure

Block d'instruction composé - instru
- instru
- instru
evaluer une circontance -
Situer ds quel instruct° on somme -

⇒ flux horizontal interaction H machine
⇒ flux vertical execut° du prog

Suite cast

Calcul propositionnel — connecteur logique

En Algo ★ Si CONDITION alors INSTRUCTION CONTROLÉE
Sinon INSTRUCTION
★ Si CONDITION alors
INSTRUCTION
INSTRUCTION
INSTRUCTION
SinonINSTRUCTION

OU disjonction
ET Conjonct°
NON négat°

Regle de De Morgan
⇔ NON (Condition 1 ou Condition 2)
⇔ NON (Condition 1) ET (Condition 2)

= NON (Condition 1 ET condition 2)
= Non (condition 1) ou (condition 2)

ALGO TABLEAU

Projet 15 jrs après la date de lancement

$x, t(x), t^2(x), t^3(x), \ldots, t^k(x)$

8, t(8), t²(8), t³(8) ... Revoir Projet
14 14 13

Exam individuel de Prog un exercice à faire.

---

déclarer un tableau d'entier de taille 10.


```
int tab[10];
```

tab ensemble de case

on accède à chaque "case" du tableau à l'aide de son nom (tab) et d'un index unique qui identifie chaque case: les index vont de 0 à n-1 la taille du tableau

affectation de la valeur 7 à la 6ème case

ECRITURE DS TABLEAU.

```
tab[5]=7;
```

lecture et affichage d'une cellule du tableau

LECTURE ET AFFICHAGE

```
printf(" %d", tab[5]);
printf(" tab[%d]= %d", 5, tab[5]);
```

Ex: Int Tab 100 cellule avec boucle 1 → 100
Aff Tab des case 100 → 1

INITIALISATION

```
int main()
int i = 0;
while (i < 100) {
    tab[i] = i+1 ;
    i++; }
```

AFFICHAGE

```
i = 99;
while (i >= 0) {
printf(" tab[%d] = %d", i, tab[i]);
```

Suite de Fibo avec Tab[100]

⚠ on n'a pas le droit d'accéder à un inexistant

unsigned fibo[100]; unsigned n = 2; unsigned = entier positif (pas négatif) de 100.

```
// initialisation avec les 2 premiers termes
fibo[0] = 0;
fibo[1] = 1;
while (n<100) {
fibo[n] = fibo[n-1] + fibo[n-2];
n++; }
```

Comment utiliser un tableau avec une fonction ?

Affiche le contenu du tableau d'entiers t de taille size

```
void print_tab (int t[], int size)
        ou (int t[100])
{
  int i = 0;
  while (i < size) {
    printf ("%d ", t[i]);
    i++; }
  printf ("\n");
}
```

Void print_tab (unsigned tab[100])

TABLEAU à DIMENSIONS


```
void test_tab_2D (){
  // declaration
  int tab [2] [3];        // Tableau 2 lignes et 3 colonnes
  // affectation
  tab [1] [1] = 5;
  printf ("%d", tab [1] [1]);
```

AFFICHER TABLEAU TABLE MULTIPLICATION

```
int mult_table [12] [12];
int i, j;

for (i = 0; i < 12; i++)
    for (j = 0; j < 12; j++)
        mult_table
```

Voir Suite

Algo rappel TABLEAU

int t[N] $\xrightarrow{en}$ C

parcours → boucle
pour i allant de 0 à (n-1) inclus faire
| ti → i+1 // afficher ti+1
fin pour

Recherche d'un élément x ds un tableau t

recherche_seq (t: tableau d'entier 10, n: entier, x: entier): booleen
données: t, tableau à parcourir
n, entier taille du tableau
x, entier element du tableau

$(t=(t_i)_i \in [0; n[$

Début
pour i allant de 0 à (n-1) inclus faire
si (ti = x) alors retourner Vrai
sinon faux // x n'existe pas ds t
fin

t	5	0	10	15	3

Recherche de l'élément max ds un tableau
valeur max (t: tableau entier, n: entier): entier
données: $t=(t_i)_{i \in [0; n-1]}$
n, entier, taille tableau
variable locale: max, entier

Début indice ← 0
max ← t0 // suppose que l'élément max est de la
pour i allant de 1 à (n-1) inclus faire 1er case
// comparer le reste des valeurs du tableau
si (ti > t indice max) alors
max ← t0 max ← i
fin pour
retourner max indice
fin

max = 5
i = 1 /
i = 2 max → 15
i = 3 max → ~~15~~
i = 4 /

a				
	4	1	2	5
	3	0	8	10
	7	13	11	9

Déclaration

$a = (a_{ij})_{i \in [0;\, l-1]}$
$\quad\quad j \in [0;\, c-1]$

l : nbr de ligne
c : nbr de colonne
accéder à un élément : a_{ij} // a[i][j] → en C

pour i allant de 0 à (l-1) faire // parcour ligne
  pour j allant de 0 à (c-1) faire // parcour colonne
    $a_{ij} \rightarrow i+j$
  fin pour
fin pour

En C.

```
// void remplissage_tab (int n, int t[])
int i;
for (i = 0; i < n; i++)
  // scanf ("%d", &t[i]);
  t[i] = i + 1; }

int main () {
  int tab [N]
  remplissage_tab_1D (tab, N);
  return 0;
```

// Affichage contenu
void affichage_tab 1D (int n, int t[])
int i;
for (i = 0; i < n; i++)
printf ("%d \t", t[i]);

// recherche element x dst
int recherche_elt (int n, int t[], int x)
// retourne 1 si vrai sinon 0
int i;
for (i = 0; i < n; i++)
if (t[i] == x) return 1; }
return 0;

// Recherche element max de t
int valeur_max (int n, int t[])
int i, max;
max = t[0]
for (i = 1; i < n; i++)
if (t[i] > max) max = t[i]; }
return max;

Rechercher un valeur dans le tableau t [6 5 3 1 2 7 2 1]

recherche
t ← 8

tque (i < n) et (ti ≤ v) faire
  si (ti = v) alors retourner i
  sinon retourner ∧
  i ← i + 1
fin tque
retourner ∧.

deb ← 0
fin ← n − 1
m ← (deb + fin) / 2

Intervertir deux cases - "Echange" "Permutation"

c ← ti
ti ← tj
tj ← c

ti ← ti + tj
tj ← ti − tj
ti ← ti − tj

t1 ← t1 + t3 ← 6
t3 ← t1 − t3 ← 5
t1 ← t1 − t3 ← 1

(avec var)

Trouver par selection - Rengen par ordre croissant

min = 1
id min = 3
permute (t, 0, 3)

[1 5 3 6 8 7 2]

min = 2
id min = 6
permute (t, 1, 6)

[1 2 3 5 8 7 6]

min = 5
id min = 6
permute (t, 3, 6)

[1 2 3 5 6 7 8]

Rechercher élément min pour [i..., n[
Tri par selection (t: tableau donnée n: entier, taille du tableau)
donné en référence t(i) tableau à trier t ∈]0; n[
var loc: i, j compteurs de boucle
id min, indice de la + petite valeur
Debut pour i allant de 0 à (n − 2) inclus faire
  id num ← i
  pour j allant de (i + 1) à (n − 1) inclus faire
    si (tj < t id min)
    alors id min ← i
  fin pr
  si (id min ≠ i) alors permute (t, i, id min)
fin pour

Tri à bulles — Compare entre 2 éléments

6	5	3	1	8	7	4

```
bulles ( t : tableau n : entier
données : n : entier, taille du tableau
données réf : t = (ti) i ∈ [0 ; n-1[
var loc : entier booléen

Début  est_trié ← faux
       tque (est_trié ≠ vrai)
            est_trié ← vrai
            pour i allant de 0 à (n-1) inclus faire
                 si (ti) t(i+1) alors
                      permute (t, i, i+1)
                      est_trié ← faux
                 fin si
            fin pour
       fin tque
```

Insertion de un trous on décale vers la droite

```
pour i allant de 1 à (n-1) inclus faire
     v ← ti   // valeur à insérer
     j ← i-1
     tque (j ≥ 0 et tj > v)
          t(j+1) ← tj
          j ← j-1
     fin
fin
```

en C

malloc sizeof

```
#define N 100

int n;
scanf("%d", &n)
int *tab = NULL
tab = (int*) malloc (n x sizeof(int))   // taille en octet
for (i=0; i<n; i++)
    tab[i] = i;
```

Dynamique

1) Déclarer et récupérer une taille par saisie utili
2) Déclarer un pointeur
3) Allouer l'espace mémoire
4) Remplissage du tableau
5) Manipulation des éléments + Affichage
6) Libération mémoire

malloc : allouer l'espace mémoire en octet du tab.
calloc : allouer l'espace mémoire en octet d'un bloc mémoire et initialise l'ensemble des cases à zero (0)
realloc : modificat° de la taille du tab.
free : libération mémoire

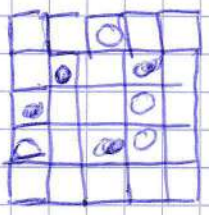
```
int *tab;
int n; // taille tab
scanf("%d", &n)

// malloc
tab = (int*) malloc (n * sizeof(int));
// manière générale : (type*) malloc (nbr d'element * sizeof(type));

// Calloc
tab(int*) calloc (n * sizeof(int)); // ensembles des cases initialisées à zero.

// realloc, l'augmentation de l'espace mémoire
tab = (int*) realloc (n/2 * sizeof(int));
```

Algo - variable aléatoire



n = 5
p = 4, 4 points

$a_{i,j} \leftarrow$ 'o' tirage aléatoire de coordonnée

→ Problème de Collision

Navette de Knuth

Résout Pbs

Pbs avec les algos qui font des tirages aléatoires.

1) Les coordonnées sont les mêmes
→ Collision = coups pour rien

2) Les biais statistiques. (sur que # les distribes soit bonnes

Navette de Knuth

$x \leftrightarrow y$
transpose x et y

Référence algorithmique effectuer pour répondre au Pbs de mélange.

1	2	3	4	5

tableau initialement ordonnée
→ But: mélanger

i et j sont des curseurs

on transpose les valeurs du curseur i avec j.
j +/- i aléatoire

point fixe. la permutat° ne change pas.

aucun point fixe acceptable.

ALGO

taille(a) = n

NavetteKnuth(a: tableau 1D de taille n): rien
donnée modifiée: t. le tableau à mélanger
Variable locales: i,j deux index

Début
| pour i allant de 0 à taille(a) - 2 faire
| | j ← alea_range(i, taille(a) - 1)
| | ai ↔ aj
| fin pour
fin

Pour Prog C –

rand() → retourne un entier tiré aléatoirement entre 0 et RAND_MAX.

x = rand()

Algo –

alea → retourne un entier entre 0 et +∞

$x \leftarrow$ alea – reçoit un entier positif.

1) Comment utiliser alea pour obtenir un tirage d'un nbr compris entre 0 et un entier n donné ?

$x \in [0; n]$

modulo –

$x \leftarrow$ alea mod$(n+1)$

2) Comment utiliser alea pour tirer aléatoirement $x \in [a, b]$

Cqfd $x \leftarrow a + (\text{alea mod } (b-a+1))$

$x \in [0, n] = x \in [a; n+a]$

$x \leftarrow a + (\text{alea mod } n+a)$

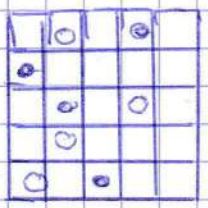
$n \Rightarrow b-a$

---

aleaRange (a,b : entier) : entier
donnée : a, b (a<b) les 2 bornes de l'intervalle de tirage
Début
retourner a + (alea mod b-a+1)
fin

Concevoir un algo qui prend en argument un tableau 2D n x n et qui positionne aléatoirement p pion noir et p pion blanc dans ce tableau (AND $2p \leq n^2$)

ex :



1 – initialise le tableau avec que des p –

InitGame (a : tableau 2D n×n ; p : entier)
Donnée modifié : a 2 plateaux à initialiser
Donnée : p : nbr de pions de chacun des joueurs
var : i, j, k, index, i', j', k'

Début pour k allant de 0 à p-1 faire
$i \leftarrow k / n$
$j \leftarrow k \cdot \text{mod } n$
$a_{i,j} \leftarrow$ '●'
fin
même pour 2 p-1

opération inverse $k \leftarrow i \times n + j$

pour k allant de 0 à n^2-2
$k' \leftarrow$ aleaRange (k, n^2-1)

i → k/n j → k mod n i' → k/n j' → k mod n $a_{i,j} \rightarrow a_{i',j'}$