I) Les instructions

Une calculatrice est une machine qui exécute les ordres qu'on lui demande !
En langage informatique, un ordre s'appelle **une instruction.**
Quand on tape une instruction sur le clavier de la machine, on la termine par la touche [Exe] ou [Enter] , et la machine exécute l'instruction qu'on lui a demandé.
Dans toute la suite, cette touche sera remplacée par le symbole ↵
Voici un exemple d'instruction 2 + 3 ↵ la machine exécute l'instruction et affiche 5

II) Les variables

L'intérieur d'une calculatrice est formé de deux parties.

Une mémoire appelée RAM (Random Access Memory) qui est composée de cases mémoire où l'utilisateur a accès (aléatoirement) et peut écrire des nombres.

Une unité centrale, qui exécute des calculs en utilisant les nombres stockés dans les cases mémoire. (l'unité centrale a besoin aussi de mémoire pour effectuer ses calculs. Elle s'appelle **la mémoire interne ou ROM** (Read Only Memory) Comme son nom l'indique, l'utilisateur peut éventuellement lire le contenu de ses cases mémoire, mais en aucun cas ne peut écrire dedans)

Pour écrire un nombre (par exemple 3) dans la RAM
il suffit de donner un nom (par exemple A) à une case de mémoire.
(la calculatrice se charge de trouver un endroit libre dans sa RAM qu'elle nommera A)
et de donner la valeur 3 à cette case mémoire A .
(chaque fois que l'utilisateur demandera un calcul avec la lettre A , la calculatrice remplacera A par le contenu de la case mémoire correspondante (ici par 3))

A s'appelle une **variable (réelle)**. En langage informatique, donner la valeur 3 à A s'écrit A prend pour valeur 3 , en abrégé on écrit **A ppv 3**

Rem : Certains noms de variable ne peuvent pas être utilisés, car le système les utilise. (Ans, Xmin, TblStart, Y_1, abs , etc) . Utiliser A , B , … , Z est sans risque.

Traduction en langage des calculatrices : A ppv 3 s'écrit $3 \longrightarrow A$
(la flèche est obtenue avec la touche [STO▶] STOre en anglais signifiant stocker
cette flèche est très parlante, elle indique qu'on met (ou stocke) 3 dans la variable A)

☞ Pour résoudre l'équation $x^2 + 2x - 3 = 0$ dans $\mathbb{R}$, écrire les instructions suivantes

$1 \longrightarrow A$ ↵ (A ppv 1 et la machine affiche 1)
$2 \longrightarrow B$ ↵ (B ppv 2 et la machine affiche 2)
$-3 \longrightarrow C$ ↵ (C ppv –3 et la machine affiche –3)
$B^2 - 4AC \longrightarrow D$ ↵ (D ppv $2^2 - 4(1)(-3)$ et la machine affiche 16)
$(-B - \sqrt{D}) / (2A)$ ↵ (la machine calcule $\dfrac{-2-\sqrt{4}}{2\times 1}$ et affiche – 3)
$(-B + \sqrt{D}) / (2A)$ ↵ (la machine calcule $\dfrac{-2+\sqrt{4}}{2\times 1}$ et affiche 1)

Rem : Tout ceci est très intéressant, mais si on veut résoudre une autre équation du second degré, il faudra réécrire ces six instructions … Est-il possible que la calculatrice garde en mémoire toute cette séquence d'instructions ? La réponse est … oui !

I) Définition d'un programme

*Un **programme** est une **liste d'instructions** écrites dans un **langage de programmation exécutable** par une calculatrice programmable. (ou un ordinateur)*
Une instruction est un ordre (ou une phrase impérative) qui effectue une action précise.
*(Un programme s'appelle aussi un **logiciel**)*

Une calculatrice programmable TI ne comprend qu'un seul langage de programmation.
(Un ordinateur peut comprendre plusieurs langages de programmation, certains étant très évolués et très proches de notre langage français)

Les langages de programmation sont généralement écrits en anglais.
Voici les traductions de quelques instructions des calculatrices TI

Entrer	Afficher	Si ... Alors ... Sinon	Fin	Etiquette	Aller à	Tant que	Pour
Input	Disp	If … Then … Else	End	Lbl	Goto	While	For

II) Comment écrire un programme ?

Pour écrire un programme, il suffit d'appuyer sur la touche PRGM
(abréviation de PRoGraM) , puis de sélectionner NEW (nouveau programme)

La calculatrice nous demande alors un nom pour notre programme.
(donnez un nom qui reflète ce que fait votre programme, par exemple EQUA2)

Sur l'écran apparaît alors PROGRAM : EQUA2
Il ne nous reste plus qu'à taper la succession d'instructions qui composent notre programme,
chaque instruction se terminant par ↵

Pour bien se repérer, la calculatrice écrira automatiquement **:** au début de chaque instruction.

Tout ce que l'on va écrire restera bien sur dans la calculatrice même si on l'éteint.

Si on veut alors voir ou modifier le programme déjà écrit, il suffira de faire PRGM
puis EDIT puis de sélectionner le nom du programme.
(**éditer** un programme, c'est voir la liste de toutes les instructions qui le composent)

III) Comment exécuter un programme ?

Lorsque notre programme sera écrit, il faudra bien demander à la calculatrice de l'**exécuter** !
Pour cela, il suffira de faire PRGM puis EXEC puis de sélectionner le nom du programme.
La calculatrice affiche alors prgmEQUA2 . Il suffira alors d'appuyer sur ↵ pour démarrer l'exécution du programme.

La calculatrice exécute dans l'ordre la 1ère instruction, puis la 2ième … jusqu'à la dernière.

IV) Comment effacer un programme ?

Pour effacer le programme TOTO , touche MEM ,
puis Mem Mgtm/Del (ou Delete),
puis Prgm puis sélectionner le programme TOTO et ↵

➥ *Quand on écrit un programme, pour trouver la liste des instructions, touche* PRGM

I) L'instruction d'affectation ⟶ (ppv)

Nous l'avons déjà vu. Ex 3 ⟶ A cad A ppv 3
On dit qu'on affecte la valeur 3 à la variable A

II) L'instruction d'affichage Disp (Display veut dire Afficher)

Cette instruction se trouve dans le menu I/O (Input / Output)) ou E/S (Entrée / Sortie)

☞ Ecrire et exécuter les programmes suivants

```
: 3 ⟶ A
: Disp 2A
: Disp 3A
```

Disp 2A La calculatrice affiche 2 fois le contenu de la variable A (6)
Disp 3A La calculatrice affiche 3 fois le contenu de la variable A (9)

```
: 3 ⟶ A
: Disp "2A"
: Disp "3A"
```

Disp "*texte*" La calculatrice affiche ce qui est écrit dans les guillemets
(essayer avec Disp " Il FAIT BEAU ")

```
: 3 ⟶ A
: Disp IL FAIT BEAU
```

La calculatrice affiche un message d'erreur car elle ne connaît pas la variable IL FAIT BEAU

```
: 3 ⟶ A
: Disp "2A =", 2A
: Disp "3A =", 3A
```

Cette instruction est très intéressante car la calculatrice affiche 2A = 6 puis 3A = 9

```
: 3 ⟶ A
: 2A
: 3A
```

On voit sur cet exemple qu'il est nécessaire dans un programme d'utiliser l'instruction Disp pour afficher les calculs intermédiaires sinon la calculatrice n'affiche que le dernier calcul demandé (sur certains modèles de TI , la machine n'affiche même rien du tout)

III) L'instruction Pause

Cette instruction se trouve dans le menu CTL (en anglais ConTroL)

A l'exécution, la calculatrice travaille vite ! Quand elle a fini, elle écrit Done (fait)
Si on veut l'arrêter pendant l'exécution d'un programme (par exemple pour lire un calcul intermédiaire) , il suffit de rajouter l'instruction **Pause** dans le programme. A l'exécution, pour finir la pause de la calculatrice, et lui demander de continuer, on appuie sur ↵

☞ Ecrire et exécuter le programme EQUA2 formé des six instructions déjà écrites feuille 1 en utilisant Disp pour afficher Delta = 16 puis $x_1 = -3$ puis $x_2 = 1$
et en utilisant Pause après l'affichage de Delta

Rem : Notre programme résout toujours la même équation $x^2 + 2x - 3 = 0$. Est-il possible d'écrire un programme qui lors de son exécution nous demande d'**Entrer** les valeurs de a , b et c puis qui résout l'équation du second degré ? La réponse est … oui !

Pour écrire un programme, il faut d'abord l'écrire en français.
La liste des instructions écrites en français s'appelle un ***algorithme****.*

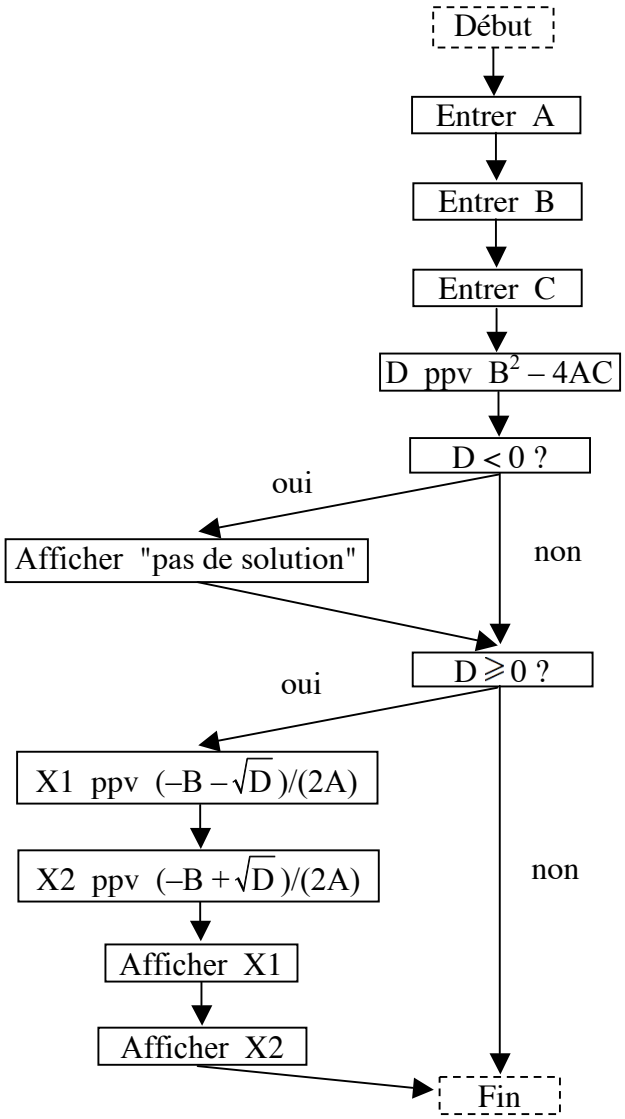
➥ Voici l'algorithme de résolution de l'équation $ax^2 + bx + c = 0$ dans $\mathbb{R}$

Entrer A
Entrer B
Entrer C
D ppv $B^2 - 4AC$
Si $D < 0$ Alors Afficher " l'équation n'a pas de solution "
Si $D \geqslant 0$ Alors X1 ppv $(-B - \sqrt{D})/(2A)$: X2 ppv $(-B + \sqrt{D})/(2A)$: Afficher X1 et X2

Rem On regroupe les cas $\Delta = 0$ et $\Delta > 0$ (car si $\Delta = 0$, on a 2 solutions égales X1 = X2)

Les algorithmes peuvent se schématiser par des ***organigrammes****.*

➥ Voici l'organigramme correspondant à notre algorithme.



Il ne nous reste plus qu'à traduire cet ***algorithme*** *(ou cet* ***organigramme*** *) en un* ***programme****, dans le* ***langage de programmation exécutable*** *par notre calculatrice TI .*

IV) L'instruction d'entrée Input (Entrer)

Cette instruction se trouve dans le menu I/O (ou E/S)

☞ Ecrire et exécuter les programmes suivants

```
: Input A
: Disp "2A"
```

A l'exécution, un ? apparaît. La calculatrice attend qu'on lui entre un nombre. Entrer un nombre suivi de ↵ La machine affiche 2A

```
: Input A
: Disp 2A
```

A l'exécution, un ? apparaît. Entrer un nombre.
La machine affiche le double du nombre entré.

```
: Input "B =", A
: Disp "2B =", 2A
```

A l'exécution, ? est remplacé par B =
Entrer un nombre. (par exemple 5) La machine affiche 2B = 10
Le nombre entré est stocké dans la variable A, bien qu'on le nomme B

V) L'instruction d'effacement d'écran ClrHome (ou EffEcr)

ClearHome ou Effacer l'écran. Cette instruction se trouve dans le menu I/O (ou E/S)
Cette instruction peut être utilisée par exemple au début d'un programme pour effacer l'écran.

VI) L'instruction conditionnelle (ou test) If . . . Then . . . End

Elle est composée de trois mots **If** . . . **Then** . . . **End** (Si . . . Alors . . . Fin)
Cette instruction se trouve dans le menu CTL
La syntaxe est la suivante

```
: If condition
: Then
: instruction
: etc
: instruction
: End
```

La calculatrice exécutera ces instructions seulement si la condition est vraie.
(Si la condition est fausse, elle ne les exécutera pas)

VII) L'instruction d'arrêt Stop

Cette instruction se trouve dans le menu CTL
L'instruction **Stop** arrête définitivement l'exécution du programme.
(Il n'est pas nécessaire de la rajouter en fin de programme.
Si D < 0 , on peut la rajouter par exemple après avoir affiché " pas de solutions ")

☞ Nous pouvons maintenant écrire le programme EQUA2 correspondant à l'organigramme de la feuille 4 . (Si la calculatrice ne comprend pas X1 et X2 , remplacer par E et F)

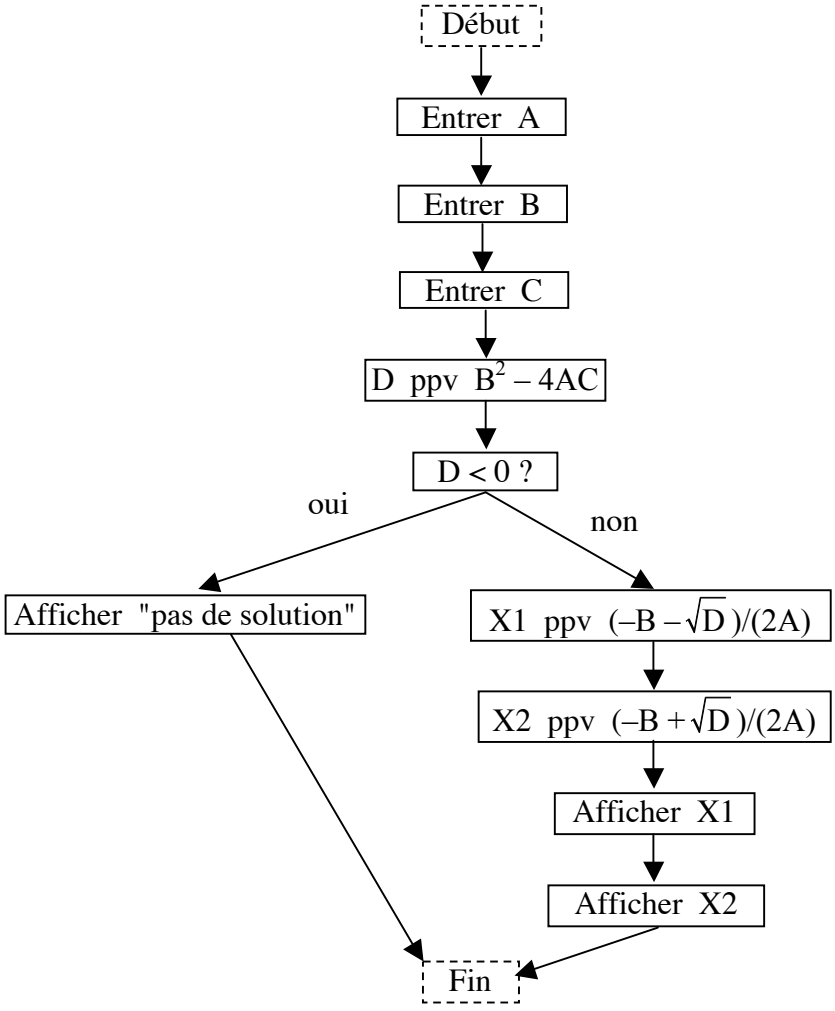
➥ *Pour insérer une instruction entre deux instructions déjà écrites,* se placer au début de la deuxième instruction et taper INS (Shift DEL) suivi de ↵
INS veut dire INSert (insérer) et DEL veut dire DELete (effacer)

On peut rajouter un **Else** (Sinon) à l'instruction conditionnelle..

: **If** *condition*
: **Then**
: *instruction* } La calculatrice exécutera ces instructions
: etc } seulement si la condition est vraie.
: *instruction* } (Si la condition est fausse, elle ne les exécutera pas)
: **Else**
: *instruction* } La calculatrice exécutera ces instructions
: etc } seulement si la condition est fausse.
: *instruction* } (Si la condition est vraie, elle ne les exécutera pas)
: **End**

☞ Réécrire le programme EQUA2
en traduisant l'organigramme suivant avec le test If … Then … Else … End

---



Rem Cela ne sert à rien d'écrire un programme si on ne vérifie pas son exécution dans tous les cas possibles.
Ce programme "marche"-t-il toujours ?

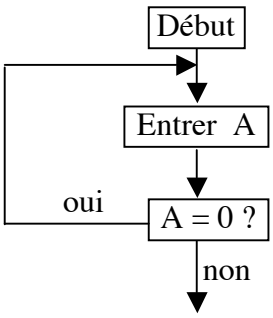
Notre programme "**tourne**" maintenant parfaitement bien, sauf si un utilisateur non averti (…) a l'idée de donner la valeur 0 au coefficient de x^2 (cad A) ! Essayez …

Dans le jargon informatique, on dit que le programme "**plante**" ou qu'il "**bugue**".

VIII) Les instructions de saut Goto et Lbl

Il faut donc traiter le cas $a = 0$, en demandant à l'utilisateur qui a entré (par mégarde) 0 pour la valeur de a , de recommencer et d'entrer une autre valeur non nulle.

Il faut donc modifier le début de notre organigramme.



Il faut donc, dans notre programme une instruction qui retourne en arrière.
Cette instruction, c'est **Goto** (Aller à)
Elle s'utilise avec **Lbl** (LaBeL veut dire Etiquette)

Il faut donc remplacer la première instruction du programme (: Input "a =", A) par les six instructions ci-contre.

```
: Lbl E
: Input "a =", A
: If A = 0
: Then
: Goto E
: End
```

☞ Réécrire le programme EQUA2 en exigeant un nombre a non nul.

Compléments

➥ *Une **commande** est une suite d'instructions séparées par : et terminée par ↵*

Au lieu d'écrire deux instructions

```
: Input "a=", A
: Disp "2a =", 2A
```

on peut écrire une commande

```
: Input "a =", A : Disp "2a =", 2A
```

Cela a un avantage. Le programme écrit est plus court.
Cela a un inconvénient. Le programme devient difficile à lire si tout est collé.

➥ *On ne peut pas modifier le nom d'un programme.*

➥ *Nous avons appris dans ces feuilles à **programmer une calculatrice**.*
Lorsque nous demandons à une calculatrice de tracer une courbe ou de nous montrer un tableau de valeurs, elle ne fait qu'exécuter des programmes déjà écrits par le fabriquant.

Nous pouvons maintenant écrire toutes sortes de programmes. Voici une petite idée ...

☞ *Ecrire un programme qui donne l'équation d'une droite qui passe par deux points donnés par leurs coordonnées et qui ne bugue jamais !!!)*

Nous allons maintenant utiliser des fonctions.
Dans cette page, nous utiliserons la fonction f définie par f (x) = 2x – 1

☞ A*ppuyer sur la touche* [Y =] *et écrire la formule* $Y_1 = 2X - 1$

X *est une variable de la RAM , cad qu'on peut lire et modifier son contenu.*

Si on écrit 2 ⟶ X ↵ la variable X aura pour valeur 2
Si on écrit Disp X la calculatrice affiche le contenu de la variable X (cad 2)

Y1 *est une variable de la ROM , cad qu'on peut lire (mais pas modifier) son contenu.*

Si on écrit 2 ⟶ Y_1 , la calculatrice répond ERROR.
Si on écrit Disp Y_1 ↵ , elle affiche l'image du contenu de la variable X par *f* (cad 3) . Chaque fois que le contenu de la variable X est modifié, la calculatrice modifie immédiatement le contenu de la variable Y_1 . Essayons pour voir …
Mais on peut écrire Y_1 ⟶ A ↵ (cela revient à recopier le contenu de Y_1 dans A)

➥ *Pour écrire* Y_1 *touche* [Y-Vars] *(ou touche* [Vars] *puis* Y-Vars *)*
(On peut utiliser un grand nombre de variables Y *(* Y_1 , Y_2 , *etc))*

Nous allons utiliser la calculatrice pour visualiser les premiers termes d'une suite récurrente, c'est-à-dire une suite définie par u_0 et u_{n+1} en fonction de u_n

Ex Soit (u_n) la suite définie par $u_0 = 2$ et pour tout n de $\mathbb{N}$ $u_{n+1} = 2u_n - 1$

☞ Taper (dans le mode de calcul normal) les successions de touches suivantes

2 ⟶ X ↵ Y_1 ⟶ X ↵ ↵ ↵ ↵

2 ⟶ X	
	2
Y_1 ⟶ X	
	3
	5
	9
	17

2 ⟶ X la machine met 2 dans X
et affiche le contenu de X (2 cad u_0)

Y_1 ⟶ X la machine calcule Y_1 (3 car X vaut 2)
et affiche le contenu de X (3 cad u_1)

En re-appuyant sur ↵ , on redemande Y_1 ⟶ X donc la machine calcule Y_1 (5 car X vaut maintenant 3) et affiche le contenu de X (5 cad u_2) et ainsi de suite …

➥ *Après avoir effectué un calcul ou exécuté une instruction ou une commande , si on re-appuie sur ↵ , la calculatrice refait le dernier calcul demandé, ou exécute la dernière instruction ou la dernière commande demandée.*

➥ *Il faut préalablement écrire dans l'éditeur de fonctions* $Y_1 = 2X - 1$
Sinon, on peut écrire $2X - 1 \longrightarrow X$
(et dans ce cas, on peut même remplacer X par une autre lettre)

➥ *On peut ainsi conjecturer les variations et les limites de suites récurrentes ..!*

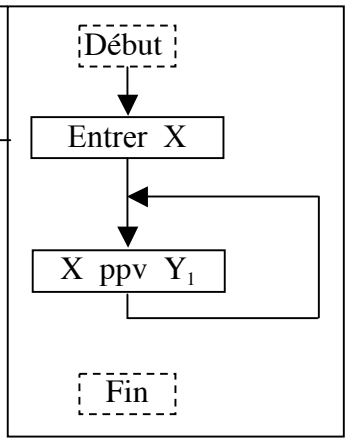
Rem : En appuyant plusieurs fois sur ↵ on ne sait plus quel terme de la suite est affiché
Est-il possible d'écrire un programme qui affiche un terme précis u_p ? La réponse est … oui !

*Une **boucle** est une instruction ou un groupe d'instructions que la calculatrice va, en exécutant le programme, répéter un certain nombre de fois.*

(nous avons déjà vu une boucle dans le programme EQUA2 . La calculatrice va répéter l'instruction Input A tant qu'on va entrer la valeur 0 dans A . Dés qu'on entre une valeur non nulle, la calculatrice va "sortir" de la boucle et continuer l'exécution du programme)

1) Une boucle peut être construite avec les instructions Lbl et Goto

☞ Ecrire le programme (qu'on appellera RECUR) correspondant à l'organigramme suivant ⟶ et exécutez-le.



La calculatrice ne s'arrêtera jamais
car nous avons créé une **boucle infinie** …

➥ *Pour arrêter un programme pendant son exécution, appuyer sur la touche ON*

La valeur de X entrée correspond à u_0

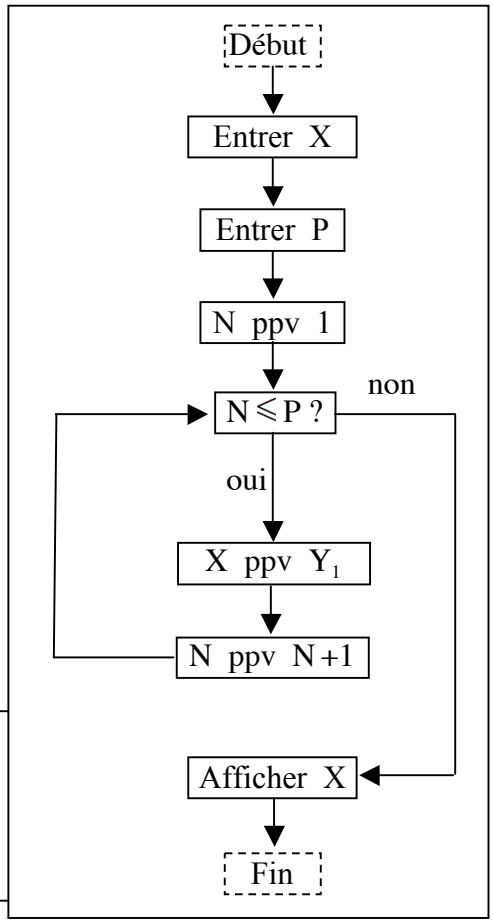
A l'exécution, nous ne voyons rien. La calculatrice travaille mais nous ne lui avons rien demander d'afficher
Rajouter une instruction d'affichage …

Maintenant la calculatrice affiche,
mais elle va trop vite !
Rajouter une Pause
pour voir les valeurs de u_0 , u_1 , u_2 etc

Pour que la calculatrice affiche u_p
il faut qu'elle exécute la boucle p fois.

Pour cela, il faut créer un compteur (une variable N)
qu'on initialisera à 1 , et qu'on incrémentera
(c'est-à-dire qu'on rajoutera 1 à son contenu)
à chaque passage dans la boucle.

Puis on "sortira" de la boucle quand le contenu
de N sera égal à p (avec un test)



☞ Ecrire le programme correspondant à l'organigramme suivant ⟶ et exécutez-le.
(il demande la valeur de u_0 et de p puis affiche u_p)

Pour que notre programme "tourne" dans tous les cas de figures, il faut aussi étudier les cas où on entre une valeur négative ou une valeur non entière pour p !

Un nombre entier est égal à sa partie entière. La partie entière d'un nombre p est notée Int(p)

(essayons dans le mode de calcul normal Int(2,35) ↵ la calculatrice répond 2 etc)
Pour trouver Int , touche [MATHS] sous-menu NUM

Juste après Input "p =", P on peut
tester si p est un nombre strictement négatif **ou** un nombre non entier, et dans ce cas afficher " p doit être un entier naturel voyons … " et faire recommencer la saisie de p
Cela s'écrit **IF** P < 0 **or** Int(P) ≠ P

(on pourrait aussi tester la condition contraire, cad si p est un nombre positif **et** un entier et dans ce cas aller à la suite du programme, sinon afficher " p doit être un entier naturel " et faire recommencer la saisie de p
Cela s'écrit **IF** P ⩾ 0 **and** Int(P) = P)

or (ou), **and** (et) sont des opérateurs booléens. Touche [TEST] puis LOGIC

☞ Réécrire le programme RECUR en exigeant un entier naturel pour p

---

2) <u>Une boucle peut aussi être construite avec l'instruction While</u>

: **While** *condition*
: *instruction*
: etc
: *instruction*
: **End**

La calculatrice exécutera ces instructions tant que la condition est vraie.
(<u>Traduction</u> **Tant que** *condition* **Faire** *instructions*)

3) <u>Une boucle peut aussi être construite avec l'instruction For</u>

: **For** (*variable* , *départ* , *arrivée* , *incrément*)
: *instruction*
: etc
: *instruction*
: **End**

La *variable* est initialisée à *départ*, et est incrémentée de la valeur de *incrément* à chaque passage dans la boucle. La calculatrice exécutera ces instructions tant que la valeur de la variable ne dépasse pas *arrivée*
(<u>Traduction</u> **Pour** *variable = départ* **jusqu'à** *arrivée* **par pas de** *incrément* **Faire** *instructions* **)**

<u>Résumé</u> Les programmes suivants sont équivalents

```
: 3 → A
: Lbl B
: If A ⩽ 7
: Then
: instructions
: A + 2 → A
: Goto B
: End
```

```
: 3 → A
: While A ⩽ 7
: instructions
: A + 2 → A
: End
```

```
: For ( A , 3 , 7 , 2 )
: instructions
: End
```

☞ Réécrire le programme RECUR en utilisant While , puis For

Solutions

PROGRAM : EQUA2

```
: Lbl E
: ClrHome
: Disp "ax² + bx + c = 0"
: Disp "a doit être non nul"
: Input "a =", A
: If A = 0
: Then
: Goto E
: End
: Input "b =", B
: Input "c =", C
```

```
: B² – 4AC ⟶ D
: Disp "Delta =", D
: Pause
: If D < 0
: Then
: Disp "pas de solution"
: Else
: (–B–√D ) / (2A) ⟶ E
: (–B+√D ) / (2A) ⟶ F
: Disp "x1 =", E
: Disp "x2 =", F
: End
```

PROGRAM : DROITE

```
: ClrHome
: Input "xA =", A
: Input "yA =", B
: Input "xB =", C
: Input "yB =", D
: Disp "équation de (AB)"
```

```
: (C – A) ⟶ E
: If E = 0
: Then
: Disp "x = c"
: Disp "c =", A
: Else
: (D – B) / E ⟶ F
: (B × C – A × D) / E ⟶ G
: Disp "y = ax + b"
: Disp "a = ", F
: Disp "b = ", G
: End
```

PROGRAM : RECUR

```
: Lbl L
: ClrHome
: Disp "suite récurrente"
: Input "u0 =", X
: Disp "calcul de up"
: Disp "p entier naturel"
: Input "p =", P
: If P < 0 or Int(P) ≠ P
: Then
: Goto L
: End
```

```
: 1 ⟶ N
: Lbl A
: If N ⩽ P
: Then
: Y1 ⟶ X
: N+1 ⟶ N
: Goto A
: End
```

```
: Disp "up =", X
```

PROGRAM : RECUR

```
: Lbl L
: ClrHome
: Disp "suite récurrente"
: Input "u0 =", X
: Disp "calcul de up"
: Disp "p entier naturel"
: Input "p =", P
: If P < 0 or Int(P) ≠ P
: Then
: Goto L
: End
```

```
: 1 ⟶ N
: While N ⩽ P
: Y1 ⟶ X
: N+1 ⟶ N
: End
```

```
: Disp "up =", X
```

PROGRAM : RECUR

```
: Lbl L
: ClrHome
: Disp "suite récurrente"
: Input "u0 =", X
: Disp "calcul de up"
: Disp "p entier naturel"
: Input "p =", P
: If P < 0 or Int(P) ≠ P
: Then
: Goto L
: End
```

```
: For ( N , 1 , P , 1 )
: Y1 ⟶ X
: End
```

```
: Disp "up =", X
```