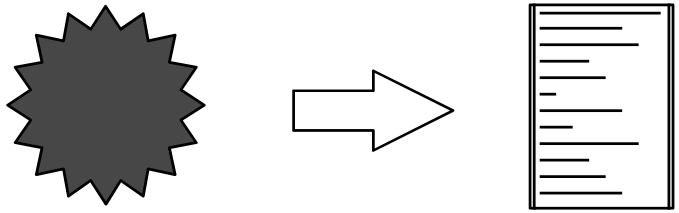
Algorithmique et programmation C++

CHAPITRE 1
Introduction
Partie 2 : notions de base de l'algorithmique

La programmation

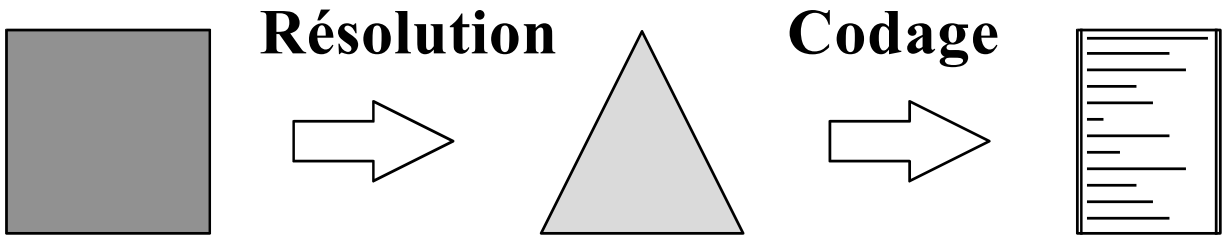


Problème **Programme**

- **Question à résoudre par une solution informatique**
- **Instance d'un problème = entrée nécessaire pour calculer une solution du problème**

- **Ensemble de données**
- **Ensemble de résultats**
  = solution informatique au problème
- **Description d' un ensemble d'actions**
- **Exécution dans un certain ordre**

Notion d'algorithme



= **Description d'un processus de résolution d'un problème <u>bien défini</u>**

= **Succession d'actions qui, agissant sur un ensemble de ressources (entrée), fourniront la solution (sortie) au problème**

- **Comment faire pour l'obtenir ?**

Pseudo code

Faire la différence entre les contraintes propres à un langage et les difficultés inhérentes à un problème donné

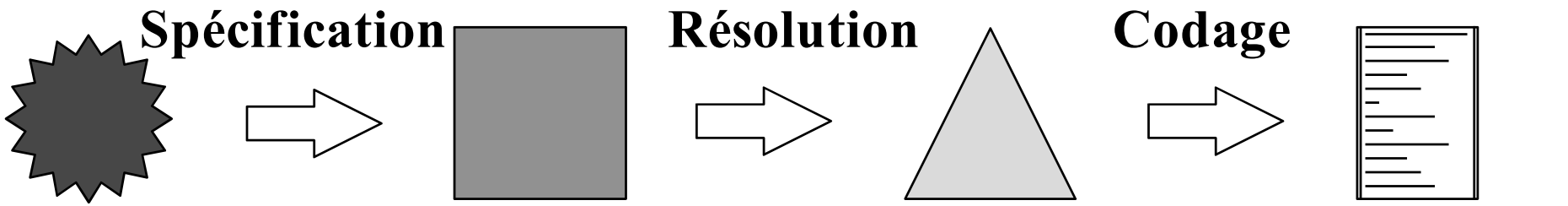
```
lire (n)
pour i ← 0 à n
  si (i mod 2) # 0
        alors afficher(i)
```

Plus abstrait, plus lisible, plus concis...

```
#include<iostream>
using namespace std;
int main ()
{
int n, i;
cin>> n;
for (i=0; i<=n; i++) {
if (i%2) {
cout <<i<<endl;
}
}
}
```

Met en avant l'essence de l'algorithme

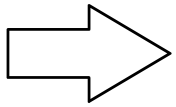
Conception d'un programme

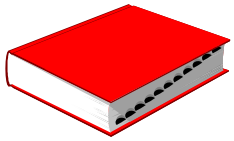


Ne pas se laisser aveugler par l'objectif final :
le codage !

Programmer, c'est communiquer

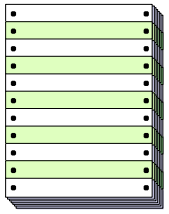
- **Avec la machine**
- **Avec soi même**
- **Avec les autres**





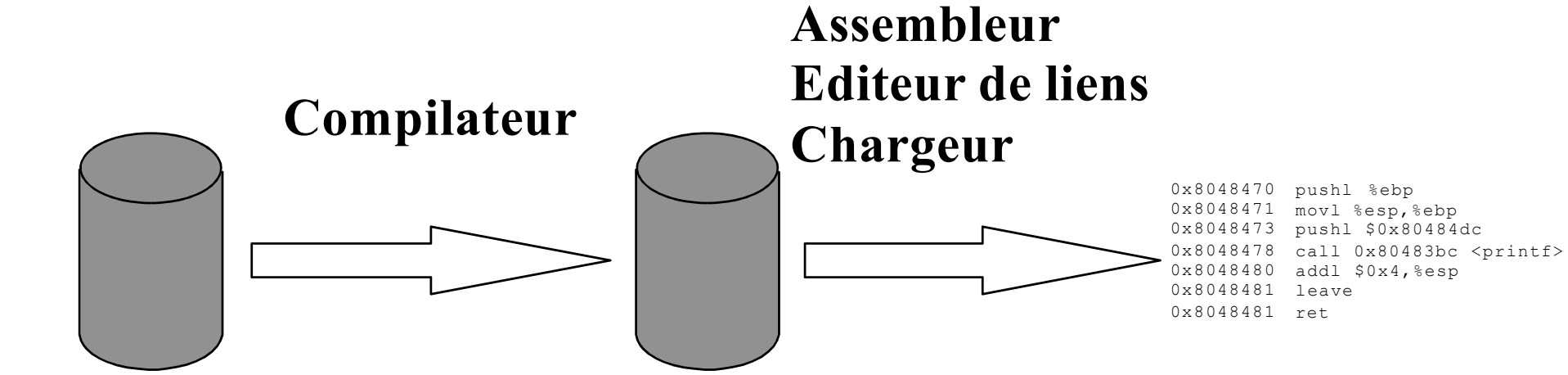
Désignations évocatrices

Algorithmes en pseudo-code concis et clairs



Programmes indentés et commentés

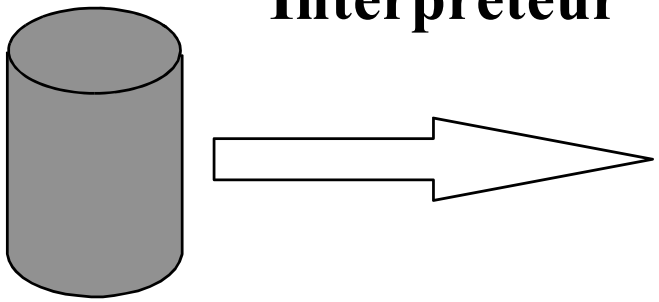
Langage compilé




```
$ OUVRIR (ou ÉDITER) monProg.cpp
$ COMPILER monProg.cpp EN monProg
$ EXÉCUTER monProg
```

Langage interprété (ex: Matlab)

Interpréteur



```
0x8048470  pushl  %ebp
0x8048471  movl %esp,%ebp
0x8048473  pushl $0x80484dc
0x8048478  call 0x80483bc <printf>
0x8048480  addl $0x4,%esp
0x8048481  leave
0x8048481  ret
```

Programme source **Code machine**

- **OUVRIR (ou ÉDITER) monProg.py**
- **INTERPRÉTER monProg.py (python monProg.py)**

Partie 1 : Concepts de base

Plan de cette partie...

- Le formalisme utilisé
- Qu’est ce qu’une variable ?
- Qu’est ce qu’un type ?
- Qu’est ce qu’une expression ?
- Qu’est ce qu’une affectation
- Les entrées / sorties ?

Formalisme...

- Un algorithme doit être lisible et compréhensible par plusieurs personnes Il dc
- donc suivre des règles
- Il est composé d'une entête et d'un corps :
  - l'entête, qui spécifie :
    - le nom de l'algorithme (**Nom :**) son
    - utilité (**Rôle :**)
    - les données "en entrée", c'est-à-dire les éléments qui sont indispensables à son bon fonctionnement (**Entrée :**)
    - les données "en sortie", c'est-à-dire les éléments calculés, produits, par l'algorithme (**Sortie :**)
    - les données locales à l'algorithmique qui lui sont indispensables (**Declaration :**)

Formalisme...

- le corps, qui est composé :
  - du mot clef **début**
  - d'une suite d'instructions **indentées**
  - du mot clef **fin**

Formalisme...

- Exemple de code :

Nom : addDeuxEntiers

Rôle : Additionner deux entiers a et b et mettre le résultat dans c

Entrée : a,b : entier

Sortie : c : entier

Déclaration : -

début

c ←a+b

fin

Qu'est ce qu'une variable...

- Une variable est une entité qui contient une information :
  - une variable possède un nom, on parle **d'identifiant**
  - une variable possède une valeur
  - une variable possède un type qui caractérise l'ensemble des valeurs que peut prendre la variable
- L'ensemble des variables sont stockées dans la mémoire de l'ordinateur

Qu'est ce qu'une variable...

- On peut faire l'analogie avec une armoire d'archive qui contiendrait des tiroirs étiquetés :
  - l'armoire serait la mémoire de l'ordinateur
  - les tiroirs seraient les variables (l'étiquette corresponderait à l'identidiant)
  - le contenu d'un tiroir serait la valeur de la variable correspondante
  - la couleur du tiroir serait le type de la variable (bleu pour les factures, rouge pour les bons de comande, etc.)

Qu'est ce qu'un type de données...

- Le type d'une variable caractérise :
  - l'ensemble des valeurs que peut prendre la variable
  - l'ensemble des actions que l'on peut effectuer sur une variable
- Lorsqu'une variable apparaît dans l'entête d'un algortithme on lui associe un type en utilisant la syntaxe suivante
  - `Identifiant de la variable :  Son type`
- Par exemple :
  - `age :  Naturel`
  - `nom :  Chaîne de Caractères`
- Une fois qu'un type de données est associé à une variable, cette variable ne peut plus en changer
- Une fois qu'un type de données est associé à une variable le contenu de cette variable doit **obligatoirement** être du même type

Qu'est ce qu'un type de données...

- Par exemple, dans l'exemple précédent on a déclaré `a` et `b` comme des entiers
  - `a` et `b` dans cet algorithme ne pourront pas stocker des réels
  - `a` et `b` dans cet algorithme ne pourront pas changer de type
- Il y a deux grandes catégories de type :
  - les types simples
  - les types complexes (que nous verrons dans la suite du cours)

Les types simples...

- Il y a deux grandes catagories de type simple :
  - Ceux dont le nombres d'éléments est fini, les **dénombrables**
  - Ceux dont le nombre d'éléments est infini, les **indénombrables**

Les types simples dénombrables...

- **booléen**, les variables ne peuvent prendre que les valeurs VRAI ou FAUX
- **intervalle**, les variables ne peuvent prendre que les valeurs entières défi nies dans cet intervalle, par exemple 1..10
- **énuméré**, les variables ne peuvent prendre que les valeurs explicitées, par exemple les jours de la semaine (du lundi au dimanche)
  - Ce sont les seuls types simples qui peuvent être défi nis par l'informaticien
- **caractères**
- Exemples :

```
masculin :  booléen
mois :  1..12
jour :  JoursDeLaSemaine
```

Cas des énumérés...

- Si l'informaticien veut utiliser des énumérés, il doit définir le type dans l'entête de l'algorithme en explicitant toutes les valeurs de ce type de la façon suivante :
  - *nom du type = {valeur1, valeur2, ..., valeurn}*
  - Par exemple :
    - JoursDeLaSemaine = {Lundi, Mardi, Mercredi, Jeudi, Vendredi, Samedi, Dimanche}

Les types simples indénombrables...

- **entier** (positifs et négatifs)
- **naturel** (entiers positifs)
- **réel**
- **chaîne de caractères**, par exemple ’cours’ ou ’algorithmique’
- Exemples :

```
age :  naturel
taille :  réel
nom :  chaîne de caractères
```

Opérateur, opérande et expression...

- Un **opérateur** est un symbole d'opération qui permet d'agir sur des variables ou de faire des "calculs"
- Un **opérande** est une entité (variable, constante ou expression) utilisée par un opérateur
- Une **expression** est une combinaison d'opérateur(s) et d'opérande(s), elle est évaluée durant l'exécution de l'algorithme, et possède une valeur (son interprétation) et un type

Opérateur, opérande et expression...

- Par exemple dans `a+b` :
  - `a` est l'opérande gauche
  - `+` est l'opérateur
  - `b` est l'operande droite
  - `a+b` est appelé une expression
  - Si par exemple `a` vaut 2 et `b` 3, l'expression `a+b` vaut 5
  - Si par exemple `a` et `b` sont des entiers, l'expression `a+b` est un entier

Opérateur...

- Un opérateur peut être unaire ou binaire :
  - **Unaire** s'il n'admet qu'un seul opérande, par exemple l'opérateur `non`
  - **Binaire** s'il admet deux opérandes, par exemple l'opérateur +
- Un opérateur est associé à *un* type de donnée et ne peut être utilisé qu'avec des variables, des constantes, ou des expressions de *ce* type
  - Par exemple l'opérateur + ne peut être utilisé qu'avec les types arithmétiques (naturel, entier et réel) ou (exclusif) le type chaîne de caractères
  - **On ne peut pas additionner un entier et un caractère**
  - Toutefois *exceptionnellement* dans certains cas on accepte d'utiliser un opérateur avec deux opérandes de types différents, c'est par exemple le cas avec les types arithmétiques (2+3.5)

Opérateur...

- La signification d'un opérateur peut changer en fonction du type des opérandes
  - Par exemple l'opérateur + avec des entiers aura pour sens l'addition, mais avec des chaînes de caractères aura pour sens la **concaténation**
    - `2+3` vaut `5`
    - `"bonjour" + " tout le monde"` vaut `"bonjour tout le monde"`

Les opérateurs booléens...

- Pour les booléens nous avons les opérateurs `non`, `et`, `ou`, `ouExclusif`
  - `non`

a	non a
Vrai	Faux
Faux	Vrai

  - `et`

a	b	a et b
Vrai	Vrai	Vrai
Vrai	Faux	Faux
Faux	Vrai	Faux
Faux	Faux	Faux

Les opérateurs booléens...

- `ou`

a	b	a ou b
Vrai	Vrai	Vrai
Vrai	Faux	Vrai
Faux	Vrai	Vrai
Faux	Faux	Faux

- `ouExclusif`

a	b	a ouExclusif b
Vrai	Vrai	Faux
Vrai	Faux	Vrai
Faux	Vrai	Vrai
Faux	Faux	Faux

Les opérateurs sur les naturels, entiers et réels...

- On retrouve tout naturellemment `+`, `-`, `/`, `*`
- Avec en plus pour les naturels et les entiers `div` et `mod`, qui permettent respectivement de calculer une division entière et le reste de cette division, par exemple :
  - `11 div 2` vaut `5`
  - `11 mod 2` vaut `1`

L’opérateur d’égalité, d’inégalité, etc...

- L’opérateur d’égalité
  - C’est l’opérateur que l’on retrouve chez tous les types simples qui permet de savoir si les deux opérandes sont égales
  - Cet opérateur est représenté par le caractère =
  - Une expression contenant cet opérateur est un booléen
- On a aussi l’opérateur d’inégalité /=
- Et pour les types possédant un ordre les opérateurs de comparaison $<, \leq, \geq, >$

Priorité des opérateurs...

- Tout comme en arithmétique les opérateurs ont des priorités
  - Par exemple * et / sont prioritaires sur + et -
- Pour les booléens, la priorité des opérateurs est `non`, `et`, `ouExclusif` et `ou`
- Pour clarifier les choses (ou pour dans certains cas supprimer toutes ambiguités) on peut utiliser des parenthèses

Actions sur les variables...

- On ne peut faire que deux choses avec une variable :
  1. Obtenir son contenu
     - Cela s'effectue simplement en nommant la variable
  2. Lui affecter un (nouveau) contenu (mettre une (nouvelle) valeur dans la variable)
     - Cela s'effectue en utilisant l'opérateur d'affectation représenté par le symbole ←
     - La syntaxe de cet opérateur est :
       - `identifiant de la variable ← expression sans operateur d'affectation`

Actions sur les variables...

- Par exemple, l'expression `c ← a + b` se comprend de la façon suivante :
  - On prend la valeur contenue dans la variable `a`
  - On prend la valeur contenue dans la variable `b`
  - On additionne ces deux valeurs
  - On met le résultat dans la variable `c`
  - Si `c` avait auparavant une valeur, cette dernière sera perdue !

Les entrées/sorties...

- Un algorithme peut avoir des interactions avec l'utilisateur
- Il peut afficher un résultat (du texte ou le contenu d'une variable) et demander à l'utilisateur de saisir une information afin de la stocker dans une variable
- En tant qu'informaticien on raisonne en se mettant **"à la place de la machine"**, donc :
  - Pour afficher une information on utilise la commande **écrire** suivie entre parenthèses de la chaîne de caractères entre guillemets et/ou des variables de type simple à affi cher séparées par des virgules, par exemple :
    - **écrire**("Le valeur de la variable a est", a)
  - Pour donner la possibilité à l'utilisateur de saisir une information on utilise la commande **lire** suivie entre parenthèses de la variable de type simple qui va recevoir la valeur saisie par l'utilisateur, par exemple :
    - **lire**(b)

Exemple d'algorithme...

Nom : euroVersDollar1

Rôle : Convertisseur des sommes en euros vers le dollar, avec saisie de la somme en euro et affichage de la somme en dollar

Entrée : -

Sortie : -

Déclaration : valeurEnEuro, valeurEnDollar, tauxConversion : Réel

début

tauxConversion ← 1.45

écrire("Votre valeur en euro :")

lire(valeurEnEuro)

valeurEnDollar ←valeurEnEuro * tauxConversion

écrire(valeurEnEuro," euros = ",valeurEnDollar,"CAN $")

fin

Exemple d'algorithme...

Nom : euroVersDollar2

Rôle : Convertisseur des sommes en euros vers le dollar

Entrée : valeurEnEuro : Réel

Sortie : valeurEnDollar : Réel

Déclaration : tauxConversion : Réel

début

tauxConversion ←1.45

valeurEnDollar ←valeurEnEuro * tauxConversion

fin

Partie 2 :
Conditionnelles et itérations

Rappels sur la logique booléenne...

- Valeurs possibles : `Vrai` ou `Faux`
- Opérateurs logiques : `non` `et` `ou`
  - optionellement `ouExclusif` mais ce n'est qu'une combinaison de `non` , `et` et `ou`
  - a `ouExclusif` b = (`non` a `et` b) ou (a `et` `non` b)
- Priorité sur les opérateurs : `non` , `et` , `ou`
- Associativité des opérateurs `et` et `ou`
  - `a et (b et c) = (a et b) et c`
- Commutativité des opérateurs et et ou
  - `a et b = b et a`
  - `a ou b = b ou a`

Rappels sur la logique booléenne...

- Distributivité des opérateurs `et` et `ou`
  - `a ou (b et c) = (a ou b) et (a ou c)`
  - `a et (b ou c) = (a et b) ou (a et c)`
- Involution
  - `non non a = a`
- Loi de Morgan
  - `non (a ou b) = non a et non b`
  - `non (a et b) = non a ou non b`

Les conditionnelles...

- Jusqu'à présent les instructions d'un algorithme étaient **toutes** interprétées **séquentiellement**
  - **Nom :** euroVersDollar2
    Rôle : Convertisseur des sommes en euros vers le dollar
    Entrée : valeurEnEuro : Réel
    Sortie : valeurEnDollar : Réel
    Déclaration : tauxConversion : Réel
    début
    tauxConversion ←1.45
    valeurEnDollar ← valeurEnEuro * tauxConversion
    fin
- Mais il se peut que l'on veuille conditionner l'exécution d'un algorithme
  - Par exemple: la résolution d'une équation du second degré est conditionnée par le signe de Δ

L'instruction `si alors sinon`...

- L'instruction `si alors sinon` permet de conditionner l'éxecution d'une suite d'instructions à la valeur d'une expression booléenne
- Sa syntaxe est :

si *expression booléenne* **alors**

suite d'instructions exécutées si l'expression est vraie

sinon

suite d'instructions exécutées si l'expression est fausse

finsi

- Le deuxième partie de l'instruction est optionnelle, on peut avoir la syntaxe suivante :

si *expression booléenne* **alors**

suite d'instructions exécutées si l'expression est vraie

fin si

Exemple...

Nom : abs

Rôle : Calcule la valeur absolue d'un entier

Entrée : unEntier : **Entier**

Sortie : laValeurAbsolue : **Entier**

Déclaration : -

début

    si unEntier ≥ 0 **alors**

        laValeurAbsolue ←unEntier

    sinon

        laValeurAbsolue ←-unEntier

    fin si

fin

Exemple...

Nom : max

Rôle : Calcule le maximum de deux entiers

Entrée : lEntier1,lEntier2 : **Entier**

Sortie : leMaximum : **Entier**

Déclaration : -

début

si lEntier1 < lEntier2 **alors**

leMaximum ←lEntier2

sinon

leMaximum ←lEntier1

fin si

fin

Exemple...

Nom : max

Rôle : Calcule le maximum de deux entiers

Entrée : lEntier1,lEntier2 : **Entier**

Sortie : leMaximum : **Entier**

Déclaration : -

début

leMaximum ←lEntier1

si lEntier2 > lEntier1 **alors**

leMaximum ←lEntier2

fin **si**

fin

Les itérations...

- Lorsque l'on veut répéter plusieurs fois un même traitement, plutôt que de copier n fois la ou les instructions, on peut demander à l'ordinateur d'éxecuter n fois un morceau de code
- Il existe deux grandes catégories d'itérations :
  - Les itérations **déterministes** : le nombre de boucles est défini à l'entrée de la boucle
  - les itérations **indéterministes** : l'exécution de la prochaine boucle est conditionnée par une expression booléenne

Les itérations déterministes...

- Il existe une seule instruction permettant de faire des boucles déterministes, c'est l'instruction `pour`
- Sa syntaxe est :

  pour *identifiant d'une variable de type scalaire* ←*valeur de début* **à** *valeur de fin*

  faire

  instructions à exécuter à chaque boucle

  fin pour
- dans ce cas la variable utilisée prend successivement les valeurs comprises entre *valeur de début* et *valeur de fin*

Exemple...

Nom : somme

Rôle : Calculer la somme des n premiers entiers positifs, s=0+1+2+... +n

Entrée : n : **Naturel**

Sortie : s : **Naturel**

Déclaration : i : **Naturel**

début

```
    s ← 0
    pour i ←0 à n faire
        s ← s+i
    fin pour
fin
```

Les itérations indéterministes...

- La boucle s'exécutera jusqu'à ce que l'expression booléenne devienne fausse.
  - L'instruction **tant que** :

    tant que *expression booléenne* **faire**

    instructions

    fin tant que

    - Ce qui signifie: "tant que l'expression booléenne est vraie, on exécute les instructions".

- Si vous ne voulez pas que votre algorithme "tourne" indéfiniment, l'expression booléenne doit faire intervenir des variables dont le contenu doit être modifié par, au moins, une des instructions du corps de la boucle

Un exemple...

Nom : invFact

Rôle : Détermine le plus grand entier e telque e! ≤ n

Entrée : n : **Naturel ≥** 1

Sortie : e : **Naturel**

Déclaration : fact : **Naturel**

début

```
    fact ← 1
    e ← 1
    tant que fact ≤ n faire
        e ← e+1
        fact ← fact*e
    fin tant que
    e ← e-1
```

fin

Explication avec n=10...

Nom : invFact
Rôle : Détermine le plus grand entier e telque e! ≤ n
Entrée : n : **Naturel**≥ 1
Sortie : e : **Naturel**
Déclaration : fact : **Naturel**
début
    fact ← 1
    e ← 1
    tant que fact ≤ n **faire**
        e ← e+1
        fact ← fact*e
    fin tant que
    e ← e-1
fin

	n	e	fact	fact ≤ n
fact ←1	10	?	1	vrai
e ←1	10	1	1	vrai
e ←e+1 fact ←fact*e	10	2	2	vrai
e ←e+1 fact ←fact*e	10	3	6	vrai
e ←e+1 fact ←fact*e	10	4	24	faux
e ←e-1	10	3	24	faux

Partie 3:
Pseudo-code schématique
(selon la norme de représentation adoptée à l'école polytechnique)

Pseudo-code schématique

- Nous décrirons les tâches à partir des opérations élémentaires suivantes:

➢ Lire	➢ Additionner
➢ Écrire	➢ Soustraire
➢ TANT QUE condition FAIRE	➢ Multiplier
➢ POUR ensemble_de_données FAIRE	➢ Diviser
➢ SI condition ALORS --- SINON	
➢ Affecter =	

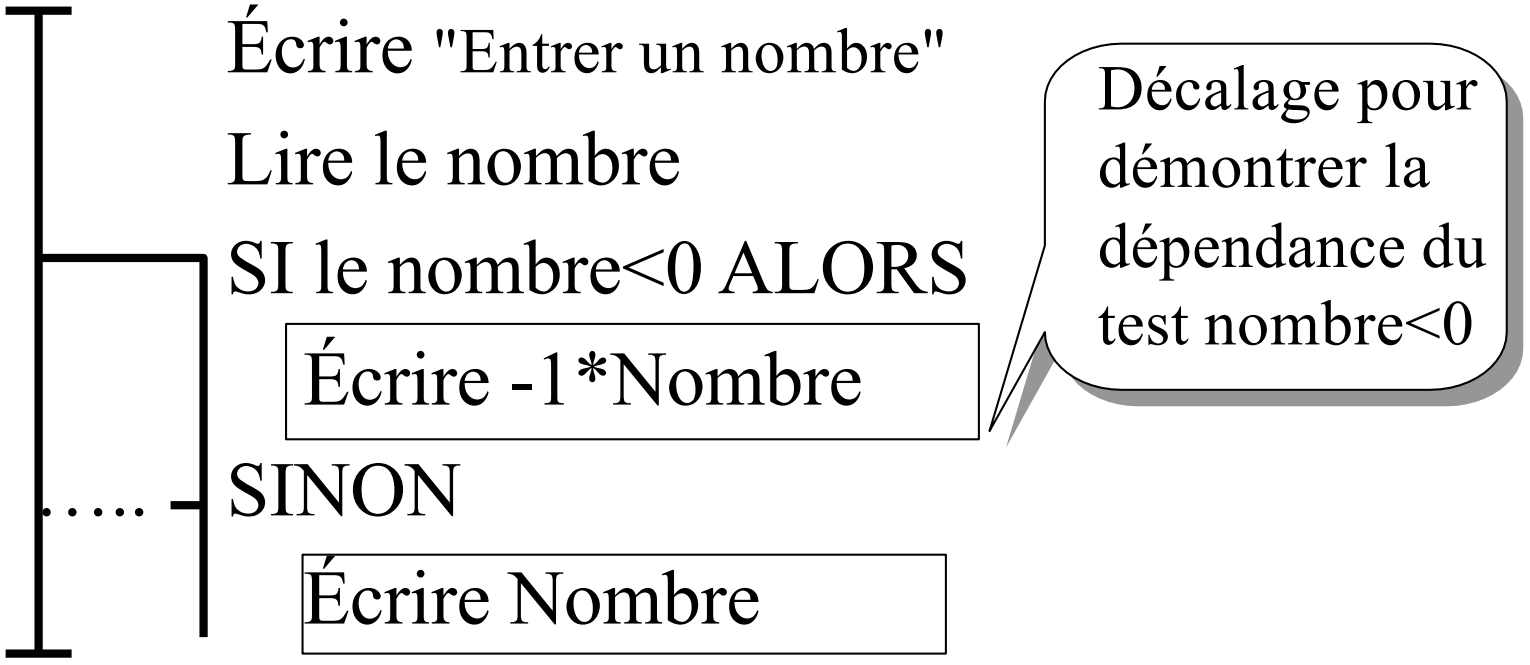
Pseudo-code schématique: Exemple 1

- Lire un nombre et écrire la valeur absolue de ce nombre
  - Quelles sont les entrées? **Un nombre.**
  - Quelles sont les sorties? **Écriture d'un nombre.**
  - Comment trouvons-nous une valeur absolue? **Il faut changer le signe du nombre s'il est négatif.**
  - Est-ce qu'il y a des répétitions ? **Non.**
  - Est-ce que des décisions ou vérifications sont requises? **Oui.** Combien? **1.** Quelles sont les alternatives? **Inversion du signe si le nombre est négatif, sinon on fait rien.** Elles influencent quelles opérations ? **Écriture du nombre.**

Pseudo-code schématique

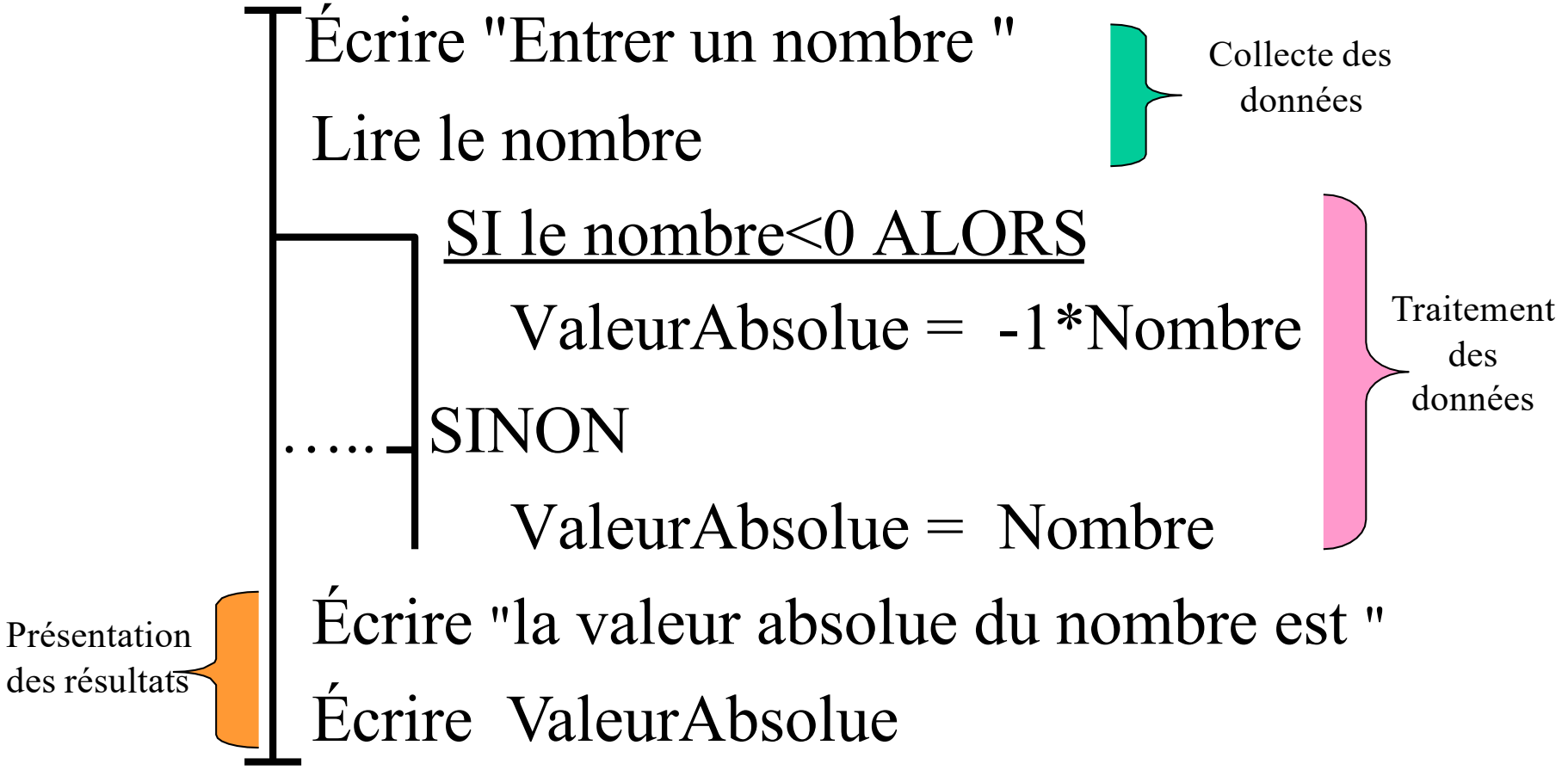
Exemple 1 - suite

- Lire un nombre et écrire la valeur absolue de ce nombre (suite)



Pseudo-code schématique

- Autre façon



Pseudo-code schématique : Exemple 2

- Deviner un nombre choisi au hasard. Indiquer si le nombre proposé est inférieur ou supérieur au nombre à deviner.
  - Quelles sont les entrées? **Un ou des nombres (dépend si devine le nombre facilement ou non).**
  - Quelles sont les sorties? **Bravo ou trop petit ou trop grand.**
  - Comment trouvons-nous un nombre choisi au hasard? **On propose des nombres tant qu'on n'a pas trouvé le bon.**
  - Est-ce qu'il y a des répétitions ? **Oui.** Combien ? **Une.** Et elles répètent quelles opérations? **Demande un nombre à l'utilisateur et vérifie ce nombre.**

Pseudo-code schématique

- Deviner un nombre choisi au hasard. Indiquer si le nombre proposé est inférieur ou supérieur au nombre à deviner.
  - Est-ce que des décisions ou vérifications sont requises? **Oui.**
  - Combien? **Deux.**
  - Quelles sont les alternatives? **Si le nombre est bien deviné, terminer le programme et écrire à l'écran bravo, ou si le nombre n'est pas deviné, vérifier le sens de l'erreur et écrire trop petit, trop grand.**
  - Elles influencent quelles opérations ? **La répétition, car recommence tant que le nombre n'est pas deviné, et aussi ce qui est écrit à l'écran (trop petit, trop grand).**

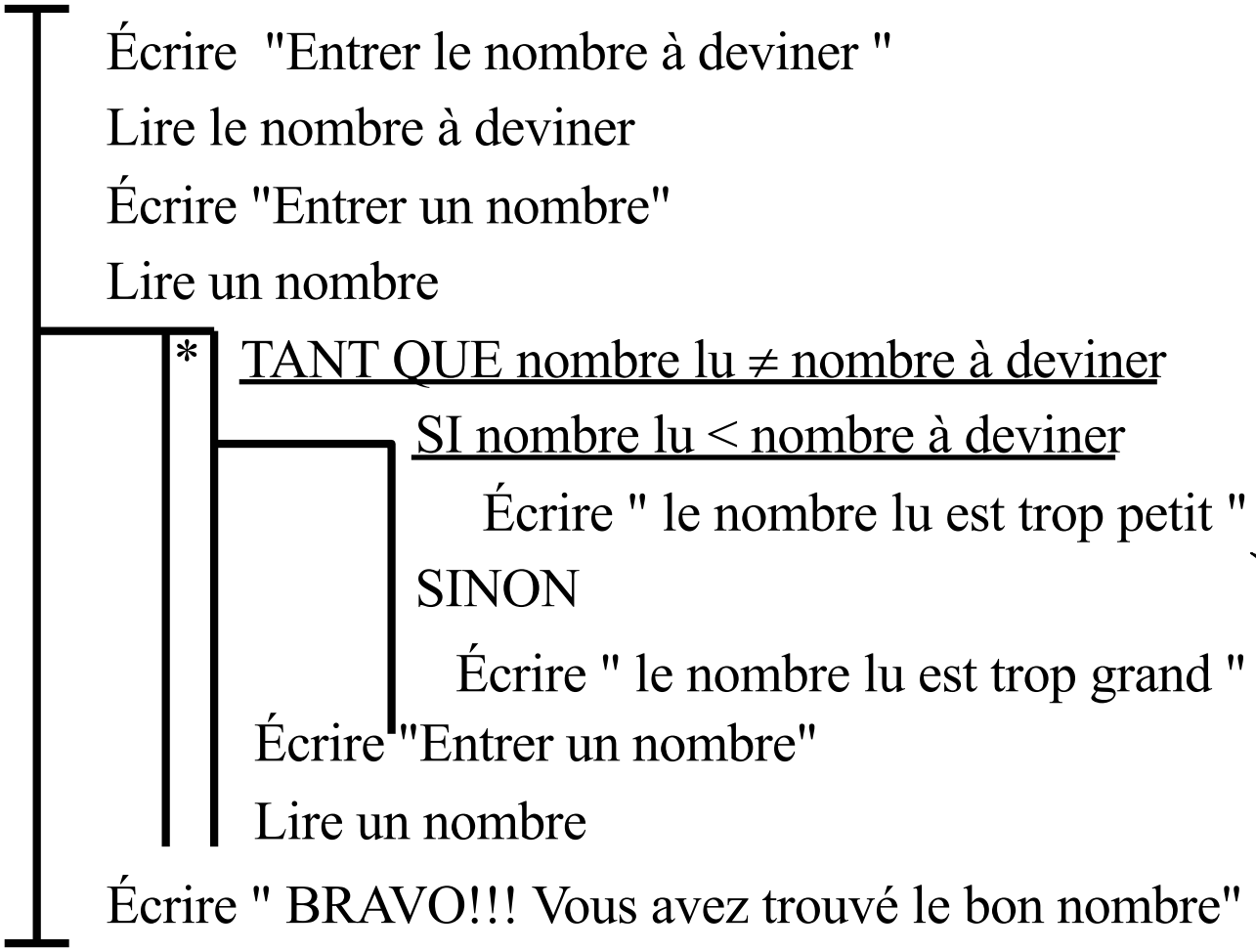
Pseudo-code schématique

- Deviner un nombre choisi au hasard (Version 1)

```
Écrire " Entrer le nombre à deviner"
Lire le nombre à deviner
Écrire " Entrer un nombre "
Lire un nombre
TANT QUE nombre ≠ nombre à deviner
    SI nombre < nombre à deviner
        Écrire "Trop petit"
    SINON
        Écrire "Trop grand"
    Afficher "Entrer un nombre"
    Lire un nombre
Écrire "BRAVO!!!"
```

Pseudo-code schématique

- Ajout du schéma répétitif



Décalage pour démontrer la dépendance du test du TANT QUE

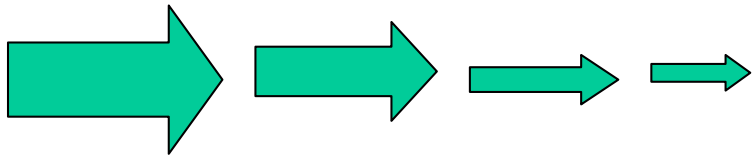
Pseudo-code schématique

- Deviner un nombre choisi au hasard (Version 2)

```
Écrire "Entrer le nombre à deviner "
Lire le nombre à deviner
Devinée = Faux
*   TANT QUE Devinée est Faux FAIRE
      Écrire "Entrer un nombre "
      Lire un nombre
      SI nombre lu = nombre à deviner ALORS
            Devinée = Vrai
.....
      SINON
            SI nombre < nombre à deviner ALORS
                  Écrire " le nombre est trop petit "
.....
            SINON
                  Écrire " le nombre est trop grand "
Écrire " BRAVO, vous avez trouvé le nombre à deviner "
```

Pensée informatique

Raffinement graduel



⇨Lors de l'élaboration d'un algorithme il est parfois utile d'envisager un processus de raffinement graduel.

⇨ Il s'agit dans un premier temps d'énumérer les opérations « évidentes » qui réalisent l'application désirée.

⇨Ces opérations évidentes ne sont pas nécessairement des opérations élémentaires.

⇨Par la suite il suffit de détailler les opérations non élémentaires à l'aide d'opérations élémentaires.

Pseudo-code schématique

- Lire trois nombres entiers quelconques et vérifier si l'un de ces nombres est le produit des 2 autres. Par exemple, si 2, 10 et 5 sont lus alors le deuxième nombre est le résultat de la multiplication des deux autres.
  - Quelles sont les entrées? **Trois nombres.**
  - Quelles sont les sorties? **Oui ou non.**
  - Comment vérifie-t'on qu'un nombre est le produit de deux autres? **On vérifie si en multipliant deux des nombres, on obtient le troisième.**
  - Est-ce qu'il y a des répétitions ? **Oui.** Combien ? **Une.** Et elles répètent quelles opérations? **La vérification du produit pour chaque combinaison.**

Pseudo-code schématique

- Lire trois nombres entiers quelconques et vérifier si l'un de ces nombres est le produit des 2 autres. Par exemple, si 2, 10 et 5 sont lus alors le deuxième nombre est le résultat de la multiplication des deux autres.
  - Est-ce que des décisions ou vérifications sont requises? **Oui.** Combien? **Une par combinaison de nombres.** Quelles sont les alternatives? **Pour chaque combinaison de nombres (3), il faut vérifier si un nombre égal le produit des deux autres.** Elles influencent quelles opérations ? **La réponse finale.**

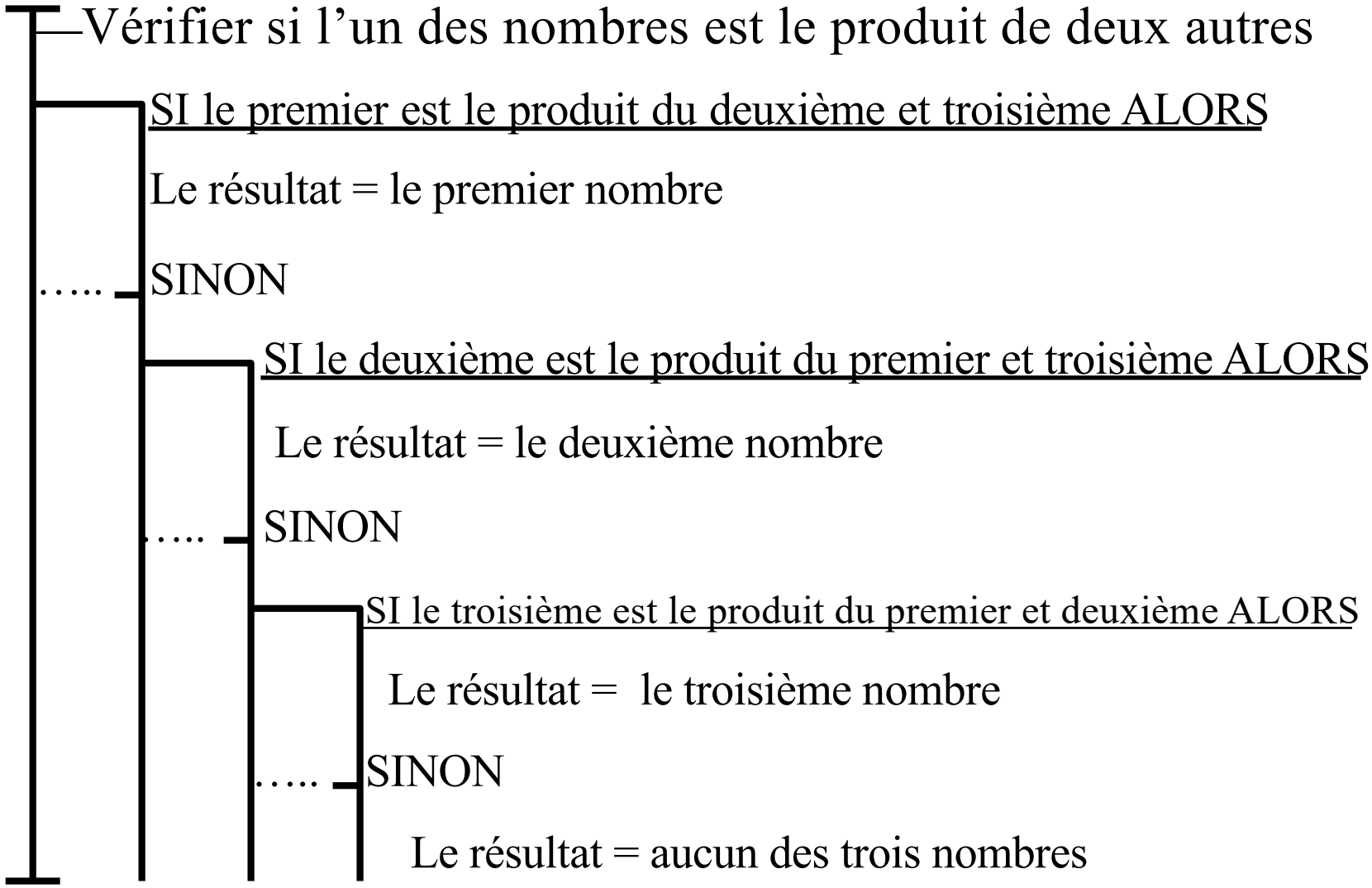
Pseudo-code schématique

- Lire trois nombres entiers quelconques et vérifier si l'un de ces nombres est le produit des 2 autres. Par exemple, si 2, 10 et 5 sont lus alors le deuxième nombre est le résultat de la multiplication des deux autres.

Écrire "Entrer trois nombres"
Lire les trois nombres
1.Vérifier si l'un des nombres est le produit de deux autres
Écrire le résultat

Opération à raffiner

Suite et fin


Vérifier si l'un des nombres est le produit de deux autres
SI le premier est le produit du deuxième et troisième ALORS
Le résultat = le premier nombre
SINON
SI le deuxième est le produit du premier et troisième ALORS
Le résultat = le deuxième nombre
SINON
SI le troisième est le produit du premier et deuxième ALORS
Le résultat = le troisième nombre
SINON
Le résultat = aucun des trois nombres