

Algorithmique et programmation structurée en C

Cours n°1



ECE PARIS · LYON
ÉCOLE D'INGÉNIEURS

Introduction

Antoine Hintzy

d'après le cours
d'Elisabeth Rendler



Plan du cours

1. [Définitions](#)
2. [Cahier des charges \(CDC\)](#)
3. [Etapes de développement d'un programme : DTI, ACD, Conception, Implémentation, Tests et Livraison](#)
4. [Analyse du CDC](#)
5. [Algorithmique : définition et exemples](#)
6. [Implémentation de l'algorithme en C](#)
7. [Correction et réutilisabilité d'un algorithme](#)
8. [Que fait-on des algorithmes ?](#)
9. [Comment concevoir un algorithme : entrées, sorties, traitements, variables \(déclaration, affectation, initialisation\) ?](#)
10. [Les variables : stockage des données, déclaration, type de variable, instructions, entrées/sorties, les tests, indentation, commentaires](#)
11. [Logigrammes : tests et boucles](#)
12. [Exemples d'algorithmes et logigrammes](#)
13. [Mon algorithme est-il correct ? jeux d'essais, preuves de terminaison, trace d'exécution, tests \(unitaires, d'intégration, de validation\)](#)
14. [Espace de travail : dossiers, fichiers](#)
15. [Environnement de développement \(IDE\) de préférence Clion : tutoriel d'installation et d'utilisation](#)
16. [Utilisation de chatGPT](#)

Définitions

Informatique : science du traitement automatique de l'information

Algorithme : suite logique et ordonnée d'instructions à suivre et d'opérations à effectuer sur des données pour obtenir un résultat

Programme informatique : ensemble d'instructions et d'opérations destinées à être exécutées par un ordinateur

Un programme est la traduction d'un ou plusieurs algorithmes.

Programme (code) source : la suite d'instructions est écrite dans un langage de programmation (langage C, python, C++, java ...)

Langage C : Inventé au début des années 1970. Un des plus utilisé, encore de nos jours. Sa syntaxe est reprise par de nombreux autres langages (C++, C#, java, javascript, php ...).

C'est un langage compilé : le code source doit être traduit en langage machine (code binaire) par un autre programme (un compilateur) avant de pouvoir être exécuté.

Programmation structurée en C : les programmes sont décomposés en sous-programmes regroupés dans des modules.

Cahier des charges (CDC)

Définition

- **Document contractuel** entre :
- un **client** (*maîtrise d'ouvrage, porteur du projet*)
- son **prestataire** (*maîtrise d'œuvre, chef de projet*)
- Formalise les **besoins, contraintes** et **objectifs** devant être respectés lors de la conception du projet.
- Les **fonctionnalités** ainsi que le **résultat** attendu y sont définis.

Cahier des charges (CDC)

Exemples simples

- **École** : l'énoncé d'un exercice ou d'un projet.
- **Société organisatrice de salons** : *Création d'une plateforme en ligne de vente et gestion de places pour des salons. Chaque visiteur pourra acheter ses tickets en ligne (le site devra être référencé), s'inscrire à des conférences et ateliers. Le staff devra pouvoir contrôler les tickets à l'entrée, placer les personnes sur leur siège, vérifier et pointer les participants à un atelier, [...]. Budget : 26000€, date butoire : 12 avril. L'interface devra être intuitive et s'utiliser sans besoin de formation.*
- **Fabricant/fournisseur** : *Création d'un support pour ordinateur portable réglable en hauteur (entre 2 et 30cm) et en inclinaison, léger, facile à ranger et à transporter. Le prix de fabrication ne devra pas excéder 5€. Les matériaux devront être solides et résistants au temps. Prévoir 3 coloris, utiliser au moins 43% de matériaux recyclés. Prévoir une housse de rangement.*

Étapes de développement d'un programme

L'objectif d'un·e développeur·se informatique  est de **créer un logiciel**, une application ou un site web **répondant à un cahier des charges** plus ou moins complet.

Il/elle peut également jouer un rôle dans l'élaboration du cahier des charges.

Étapes de développement d'un programme

1. (*facultatif*) *Participation à la rédaction du cahier des charges*

2. Analyse du cahier des charges

Extraction des informations utiles : méthode DTI (*Données, Traitements, Interfaces*), ACD (Analyse Chronologique Descendante).

3. Conception

Création des algorithmes/logigrammes à partir des données et traitements identifiés, ainsi que des interfaces Homme/machine (IHM).

4. Implémentation

Retranscription des algorithmes dans un langage de programmation. Choix des structures de données. Optimisations.

5. Tests

6. Livraison

Analyse du CDC

Méthode Données Traitements Interfaces (DTI)

◎ Données

Identification des données (noms) que l'on va traiter.

Ex : dans un logiciel de gestion des utilisateurs, on aura le nom, le prénom, la date de naissance (jour, mois, année)...

◎ Traitements

Identification et organisation des traitements (verbes).

Ex : inscription d'un nouvel utilisateur.

◎ Interfaces (Homme/machine (IHM))

Obtention des données (via le clavier, la souris...) et affichage des résultats (à l'écran).

Ex : formulaire de création d'un nouvel utilisateur avec 2 champs de texte (nom), 3 listes déroulantes (date), et un bouton (créer).

Analyse du CDC

Analyse Chronologique Descendante (ACD)

Vise à identifier et ordonner de façon chronologique les traitements de notre application.

On part d'une description générale de ce que l'on veut faire (la tâche/le traitement à réaliser), puis on la décompose en plusieurs sousparties, qu'on décompose elles aussi en sous-parties jusqu'à obtenir des sous-tâches très simples. L'objectif étant notamment d'identifier les sous-tâches utilisées à plusieurs endroits.

Exemple :

Tâche : Inscription d'un utilisateur.

Décomposition : Récupération du prénom, récupération du nom, sauvegarde du nouvel utilisateur.

Décomposition de la sous-tâche "Récupération du prénom" : Demander de saisir le prénom, récupérer le prénom au clavier (texte). **Etc.**

Commençons par le commencement...

L'algorithme

Photo : © [Ryan Quintal / Unsplash](#)

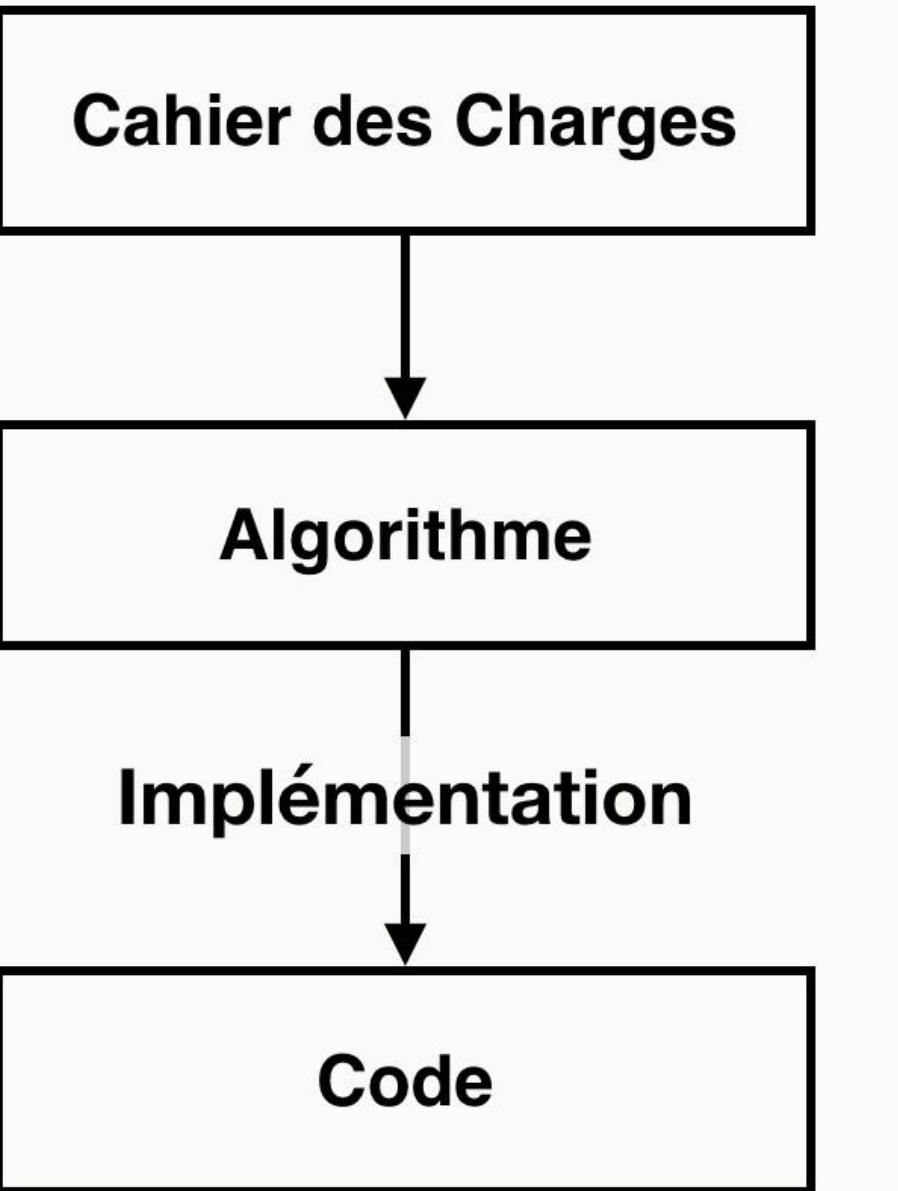


L'algorithme

Définition

Suite définie et claire d'instructions permettant de résoudre un problème.





L'algorithme

sans nécessiter de connaissances dans un langage de programmation en particulier.

On les écrit donc dans une langue naturelle (= *non informatique, le français ou l'anglais par exemple*). On peut également les représenter sous la forme de schémas (ex : logigrammes) ou de pseudo-code.



L'algorithmatique

Exemple : Recette de cuisine !

- **Données en entrée :** les ingrédients. 🥬🥒🍅🧅
- **Algorithme :** instructions à effectuer dans l'ordre décrit. 📖
(mélanger, chauffer, découper, faire bouillir, laisser reposer X heures, répéter pour chaque X...)



L'algorithme

- **Résultat :** un plat succulent, prêt à être mangé  .

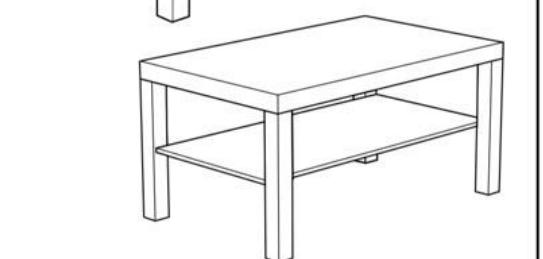
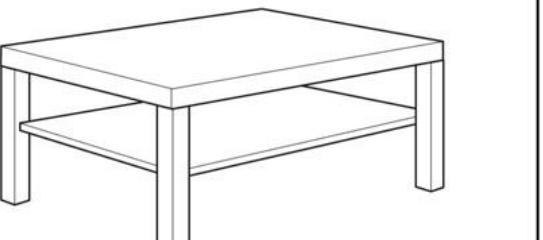


L'algorithme

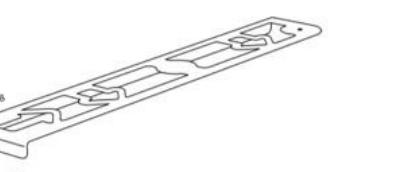
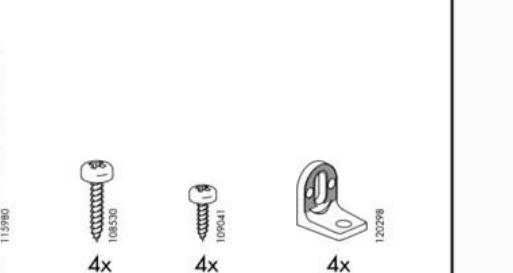
Exemple : Notice IKEA 



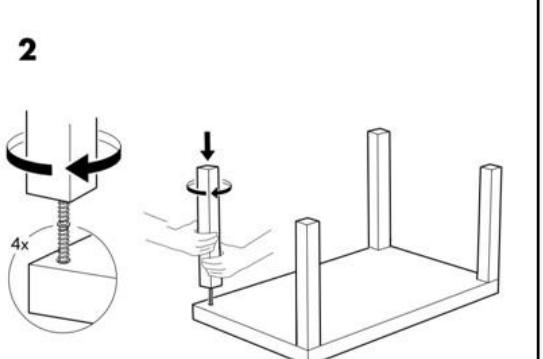
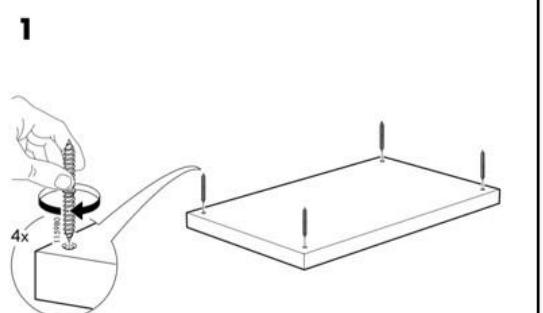
LACK



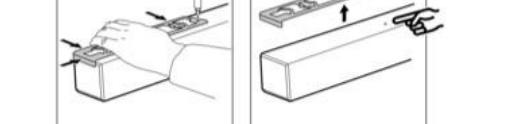
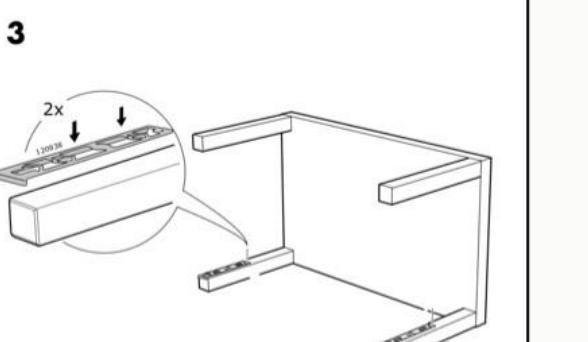
IKEA
Design and Quality
IKEA of Sweden



3



AA-472482-2



5



L'algorithme

notices de montage, IKEA a fait le choix de schémas sans texte, les rendant par la même occasion, beaucoup plus claires. *Normalement...*

On voit bien : le résultat final, les prérequis et les instructions successives.

Crédits : IKEA https://www.ikea.com/fr/fr/assembly_instructions/lack-tablebasse_AA-472482-3_pub.pdf

Exemple d'algorithme informatique (en français 🇫🇷)

DÉBUT



L'algorithme

```
ENTIER n ← 0
ÉCRIRE "Entrez un nombre :"
LIRE n
SI n >= 0 ALORS
    ÉCRIRE n + " est positif ou
nul."
SINON
    ÉCRIRE n + " est négatif."
FIN SI
FIN
```



L'algorithme

En informatique, les algorithmes suivent des règles syntaxiques les rendant compréhensibles rapidement par les développeur.se.s. Ils sont généralement écrits en anglais.

Implémentation

Les algorithmes en informatique ont pour vocation d'être transposés dans un ou plusieurs langages de programmation.



L'algorithmatique

On parle alors d'**implémentation**
d'un algorithme.

Exemple : *Implémentation en langage C de tel algorithme.*



L'algorithme

Exemple d'algorithme informatique (fr)

Revenons sur l'algorithme vu précédemment :

DÉBUT

ENTIER n \leftarrow 0

ÉCRIRE "Entrez un nombre :"

LIRE n

SI n \geq 0 ALORS

 ÉCRIRE n + " est positif ou nul."

SINON

 ÉCRIRE n + " est négatif."

FIN SI FIN



L'algorithme

Implémentation de l'algorithme précédent, en C

```
#include <stdio.h>
int main(void)
{
    int n = 0;

    printf("Entrez un nombre :\n");
    scanf("%d", &n);

    if(n >= 0)
    {
        printf("%d est positif ou nul.", n);
    }
    else
    {
        printf("%d est négatif.", n);
    }

    return 0;
}
```



L'algorithme

✓ Correction ✓

Un algorithme est **correct** si pour chaque variation de données qu'on lui donne en entrée, il se termine avec un ou le résultat correct.

Exemple :

Si notre algorithme est censé donner la somme de deux entiers qu'on lui fournit, alors en lui transmettant les entiers 3 et 5, il doit toujours donner le résultat 8, et uniquement celui-ci.



L'algorithme



L'algorithme

➡ Réutilisabilité!

Un algorithme est réutilisable s'il est suffisamment paramétrable en entrée pour pouvoir être utilisé dans plusieurs cas de figure.

Exemples :

- Une calculatrice avec comme uniques boutons [1+3] et [=] est moins réutilisable qu'une calculatrice avec tous



L'algorithme

les chiffres ([0], [1], ..., [9]) et tous les opérateurs ([+], [-], [=] ...).

- Un algorithme qui dit "Bonjour Toto" est moins réutilisable qu'un algorithme nécessitant qu'on lui fournisse un prénom, avant de lui dire bonjour. Car le premier devrait être dupliqué pour dire bonjour à "Tata" par exemple.



L'algorithme

Quand fait-on des algorithmes ?

-  **Débutants** : pour développer la logique informatique, sans pour autant avoir de connaissances dans un langage.
-  **Professionnels** : quand un problème est compliqué à résoudre et nécessite une réflexion avancée.



L'algorithme

Le reste du temps, ils réfléchissent mentalement à l'algorithme en même temps qu'ils codent.

- ◎  **Experts/chercheurs** : l'une de leurs tâches peut être d'optimiser des algorithmes existants pour les rendre moins complexes ou d'en créer de nouveaux. Ils peuvent ou non les rendre publiques.



L'algorithme

Vocabulaire : Algorithme naïf

Un algorithme écrit instinctivement, sans recherches ni réflexion préalable, est appelé **algorithme naïf**.

D'autres algorithmes, ayant fait l'objet de recherches et d'optimisations, offrent de bien meilleurs résultats (en termes de complexité). Ils peuvent être trouvés sur le web, dans des livres spécialisés...



L'algorithme

Vocabulaire : Algorithme

glouton/gourmand/goulu 😊 (🇬🇧
greedy algorithm) Un **algorithme glouton** est un algorithme qui suit le principe de faire, étape par étape, un choix optimum local, dans l'espoir d'obtenir un résultat optimum global. *Source : [Wikipedia](#)*



Voyons comment concevoir un algorithme...

Méthodologie

1. Identifier les (potentielles) entrées

De quelle données notre algorithme a-t-il besoin pour travailler ?

2. Identifier les (potentielles) sorties

D'après les données reçues en entrée, quel résultat notre algorithme doit-il retourner ?

3. Identifier et organiser les étapes/traitements

Comment pouvons-nous y parvenir ?

Les variables

Stockage des données

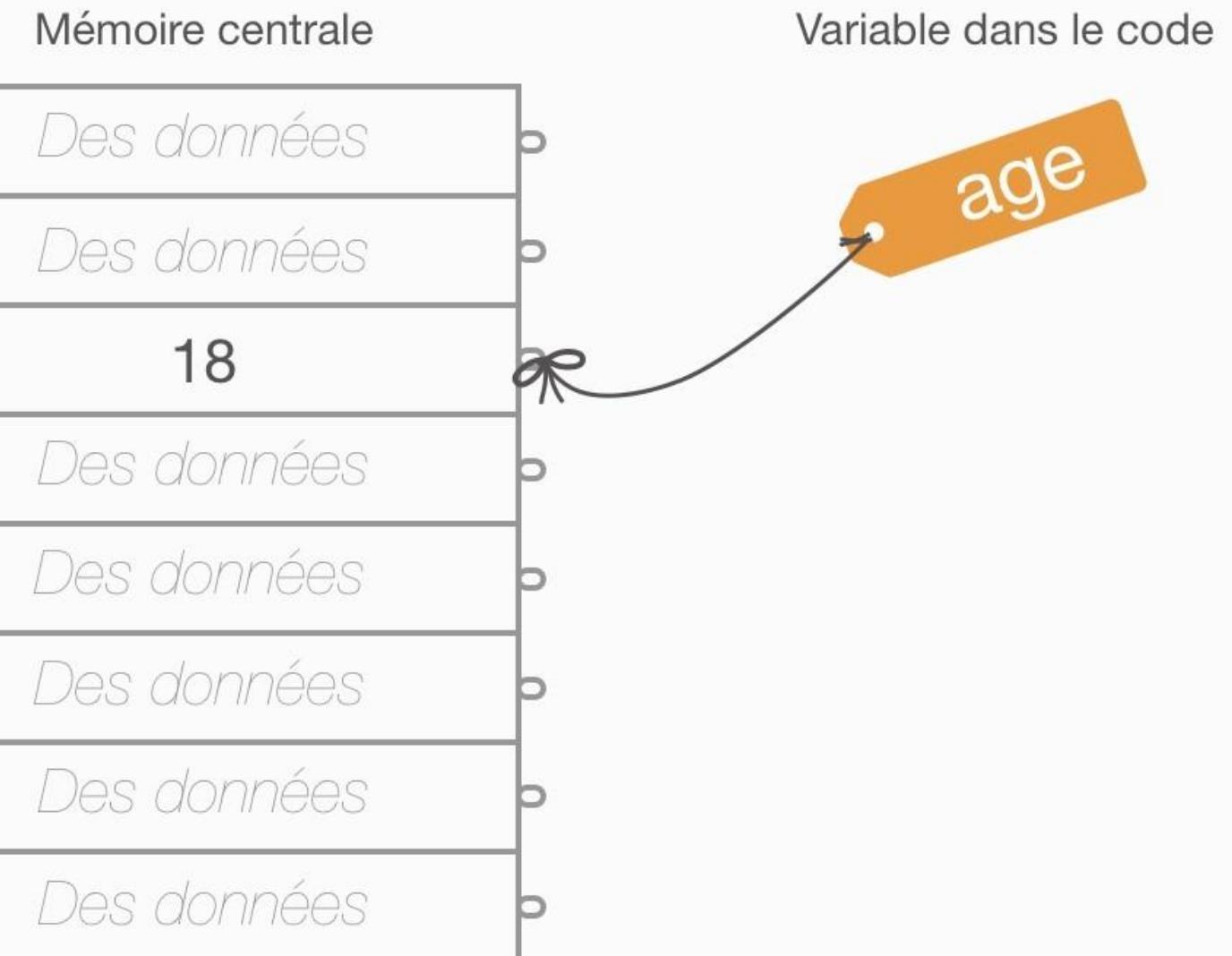
Les programmes informatiques traitent des données, enregistrées dans la mémoire temporaire de l'ordinateur (*mémoire centrale*).

Nous accédons à ces espaces mémoire depuis notre code au moyen de **variables**.

Une variable possède un **nom**, et est rattachée à une donnée stockée en mémoire. La valeur de cette donnée peut varier au cours de l'exécution de notre programme (d'où le nom de "variable").

Dans l'exemple ci-contre, la variable "**age**" contient la valeur **18**.

NB : Sauf exceptions, deux variables ne peuvent pas avoir le même nom.



Les variables

Les variables - Déclaration

Pour que l'ordinateur sache qu'il doit réserver un espace mémoire pour une variable de notre programme, nous devons **déclarer cette variable**.

C'est lors de la déclaration de la variable qu'on lui donne son **nom**. Il doit être unique, sauf dans certains cas que nous verrons plus tard.

Déclaration d'une variable en C :

nom du type nom_de_la_variable ;

- Pas d'accents, pas d'espaces, pas de caractères spéciaux ou de ponctuation. Pas de mots-clés réservés du langage.
- Les noms commencent par une minuscule
- Donner un nom qui a un sens

Les variables

NB : Le type d'une variable détermine la taille (en octets) qu'elle occupe en mémoire.

type	données	taille en octets	plage de valeurs
<i>(signed) char</i>	petits entiers naturels (ou caractères)	1	
<i>unsigned char</i>	petits entiers relatifs (ou caractères)	1	
<i>(signed) short int</i>	entiers courts	2	
<i>unsigned short int</i>	entiers courts non signés	2	
<i>(signed) int</i>	entiers	4	
<i>unsigned int</i>	entiers non signés	4	
<i>(signed) long long int</i>	entiers longs	8	
<i>unsigned long long int</i>	entiers longs non signés	8	
<i>float</i>	réels	4	3.4×10^{-38} à 3.4×10^{38} (valeurs absolues)
<i>double</i>	réels	8	1.7×10^{-308} à 1.7×10^{308} (valeurs absolues)
<i>long double</i>	réels	12	

Les variables



Les variables - Affection

L'**affectation** de la valeur x à la variable – *de nom* – "v" revient à stocker la valeur x dans l'espace mémoire de "v".

On dit que v prend la valeur x .

Affectation en C avec l'opérateur =

Exemple : v = x ;

Les variables



Les variables - Initialisation

L'initialisation d'une variable est la combinaison de sa déclaration avec sa 1^{ère} affectation. La variable se voit alors attribuer une valeur dès sa déclaration.

Les variables

Types de variables

En algorithmique, lors de la déclaration d'une variable, on peut préciser un **type** : **ENTIER, DECIMAL, CARACTÈRE...** :

◎ ENTIER age

Pour certains langages informatiques tel que le C, préciser le type des variables est obligatoire pour indiquer à l'ordinateur quelle quantité d'espace mémoire réservé pour stocker la valeur de la variable. Pour d'autres langages tel que le Javascript, il n'est pas nécessaire de préciser le type (il est déduit), nous pouvons dans ce cas déclarer une variable dans un algorithme comme ceci :

◎ VARIABLE n

Les instructions

Les instructions de base en algorithmique

Instruction	Description
DÉBUT / FIN	Début / fin de l'algorithme.
ENTIER age	Déclaration d'une variable pouvant contenir un entier, nommée "age". ⚠️ Préférez l'initialisation car ici, age prend une valeur aléatoire.
compteur ← 2	Affectation de la valeur 2 à la variable compteur . compteur vaut maintenant 2 — ⚠️ La variable doit être déclarée au préalable.
ENTIER compteur ← 0	Initialisation d'une variable pouvant contenir un entier, nommée " compteur ", de valeur initiale "0" — ✅ Il est recommandé de toujours initialiser ses variables.
ENTIER age , compteur ← 3	Déclaration/Initialisation de plusieurs variables de même type possible grâce à une ENTIER année virgule ou via une autre instruction (obligatoire si le type est différent).
VARIABLE prenom	NB : ici, seule la variable " compteur " est initialisée , avec la valeur "3".

Les instructions

Les entrées/sorties

Instruction	Description
ÉCRIRE "Bonjour "	Affichage à l'écran (dans la console) de : Bonjour
ÉCRIRE "Le comteur indique " + compteur	Le comteur indique 3 (si la variable compteur vaut 3).
LIRE age	Demande une valeur à l'utilisateur (généralement au clavier), qui sera stockée dans la variable " age ".

Les instructions

Les entrées/sorties en langage C

Afficher la valeur d'une variable (sortie) : Pour afficher la valeur d'une variable, on utilise la fonction **printf** de la bibliothèque stdio.h.

Pour chaque variable à afficher, il faut indiquer un format. Le format est indiqué par le symbole % suivi d'une ou deux lettres qui dépendent du type de la variable.

type	format
<i>char</i>	%c
<i>short int, int</i>	%d
<i>long long int</i>	%lld
<i>float</i>	%f %e (notation scientifique)
<i>double, long double</i>	%Lf

Les instructions

Les entrées/sorties en langage C

Exemples de code en C et d'exécutables avec printf :

```
#include <stdio.h>

int main() {
    int x=18;
    printf("%d\n",x);
    printf("la valeur de x est : %d\n",x);
    printf("le double de %d est : %d\n",x,2*x);
    return 0;
}
```

```
D:\Cours\2023-2024\Cours1\premierProg
18
la valeur de x est : 18
le double de 18 est : 36
Process finished with exit code 0
```

```
#include <stdio.h>

int main() {
    float x;
    x = 43.148;
    printf("%6.3f\n",x);
    printf("%10.4f\n",x);
    printf("%10.3f\n",x);
    printf("%6.1f\n",x);
    printf("%06.2f\n",x);
    return 0;
}
```

Le programme ci-dessus affiche :

```
D:\Cours\2023-2024\Cours1\premierProg
43.148
43.1480
43.148
43.1
043.15
Process finished with exit code 0
```

Les instructions

Les entrées/sorties en langage C

Récupérer une valeur saisie au clavier (entrée) : pour récupérer des valeurs saisies au clavier, on peut utiliser la fonction **scanf** de la bibliothèque stdio.h.

Il faut :

- Inclure la bibliothèque (directive de pré-compilation)
- Appeler la fonction *scanf* « au bon moment » dans le programme (appel de fonction)

```
/*inclusion des bibliothèques*/
//pour pouvoir utiliser les fonctions de la bibliothèque stdio
#include <stdio.h>

/*Sous-programme principal : point d'entrée du programme*/
int main()
{
    //déclarations des variables
    int a;
    float x,y;
    //affichage d'un message pour demander la saisie d'un entier
    printf("saisir un entier :\n");
    //récupération de la valeur saisie (entier) et stockage dans la variable a
    scanf("%d", &a); ←
    //affichage d'un message pour demander la saisie de deux réels
    printf("saisir deux réels (séparés par un espace) :\n");
    //récupération des valeurs saisies (réelles) et stockage dans les variables x et y
    scanf("%f %f", &x, &y); ←

    /*suite du programme :
    on peut désormais utiliser les valeurs saisies qui ont été stockées dans les variables*/
    printf("\n%d*%.2f = %.2f\n", a, x, a*x);
    printf("%.2f+%.2f = %.2f\n", x, y, x+y);
    return 0;
}
```

Les instructions

Les entrées/sorties en langage C

- Les formats des valeurs à lire au clavier sont indiqués entre guillemets doubles
- On peut indiquer plusieurs formats.
- Pour chaque format, il faudra ensuite indiquer l'adresse d'une variable dans laquelle la valeur lue au clavier sera stockée.
- Pour indiquer l'adresse d'une variable on utilise l'**opérateur d'adressage &**
L'adresse d'une variable se note **&nom_de_la_variable**
- Les adresses des variables sont indiquées après les formats, chacune séparée par une virgule.

```
scanf ("%f %f", &x, &y);
```

2 formats : 2 adresses de variables
La première valeur saisie sera stockée dans la première variable (x), la deuxième valeur saisie dans la deuxième variable (y)

Les instructions

Les tests

- ◎ **SI** *Condition* **ALORS**

Bloc d'instructions exécuté si la Condition est vraie

FIN SI

- ◎ **SI** *Condition* **ALORS**

Bloc d'instructions exécuté si la Condition est vraie

SINON

Bloc d'instructions exécuté si la Condition n'est pas vraie

FIN SI

NB : remplacer le *texte gris en italique* par ce qui est indiqué. — "Bloc d'instructions" = une ou plusieurs instructions (une par ligne)

Les instructions

Les tests

- ◎ **SI** *Condition1 ALORS*

Bloc d'instructions exécuté si la Condition1 est vraie

SINON SI *Condition2 ALORS*

Bloc d'instructions exécuté si toutes les conditions qui précédent la Condition2 sont fausses mais que la Condition2 est vraie

SINON

Bloc d'instructions exécuté si aucune des conditions précédentes n'est vraie

FIN SI

NB : il peut y avoir autant de **SINON SI ... ALORS ...** que l'on souhaite, mais un seul **SINON** (toujours à la fin).

Un seul *Bloc d'instructions* est traité (ex : si la *Condition1* et la *Condition2* sont vérifiées (vraies), seul le premier *Bloc d'instructions* sera exécuté.

Les instructions

Les boucles

- ◎ **POUR** variable \leftarrow valeur1 **À** valeur2 **PAR PAS DE** incrément **FAIRE**

Bloc d'instructions répété ((valeur2 – valeur1) / incrément + 1) fois

*Exemple : avec valeur1 = 2, valeur2 = 5 et incrément = 2 (généralement 1),
le bloc d'instructions va être exécuté 2 fois avec variable valant 2, puis 4.*

FIN POUR

"**PAR PAS DE** incrément" facultatif. Par défaut : 1.

- ◎ **RÉPÉTER** n **FOIS**

Bloc d'instructions répété n fois

FIN RÉPÉTER

Les instructions

Les boucles

- **TANT QUE** *Condition FAIRE*

Bloc d'instructions répété tant que la Condition est vraie

FIN TANT QUE

- **FAIRE**

Bloc d'instructions exécuté au moins une fois

(même si la Condition est fausse)

puis répété tant que la Condition est vraie

TANT QUE *Condition*

Indentation

L'indentation consiste à décaler de n tabulation·s ou n espace·s vers la droite le bloc d'instructions sous-jacent.

Exemple : tout ce qui se trouve dans un test **SI** est indenté d'un niveau supplémentaire vers la droite.

 Non indenté

SI score > 1000 ALORS

ÉCRIRE "Nouveau record."

FIN SI

 Indenté

SI score > 1000 ALORS

ÉCRIRE "Nouveau record."

FIN SI

On voit clairement que l'instruction **ÉCRIRE** est dans le test **SI**.

Commentaires

Les commentaires issus de l'ACD sont indispensables pour mieux comprendre le code qui est déduit de l'algorithme. **Sans ces commentaires, il est impossible de comprendre le code pour tout personne extérieure et même pour celui qui l'a implémenté dans une durée ultérieure. Les commentaires ont autant de valeur que le code lui-même sinon plus.**

Exemple d'algorithme

DÉBUT

ENTIER nombre \leftarrow 0

ÉCRIRE "Saisissez un nombre au clavier :"

LIRE nombre

SI nombre $\% 2 == 0$ ALORS

 ÉCRIRE "Ce nombre est pair."

SINON

 ÉCRIRE "Ce nombre est impair."

FIN SI

FIN

Logigramme

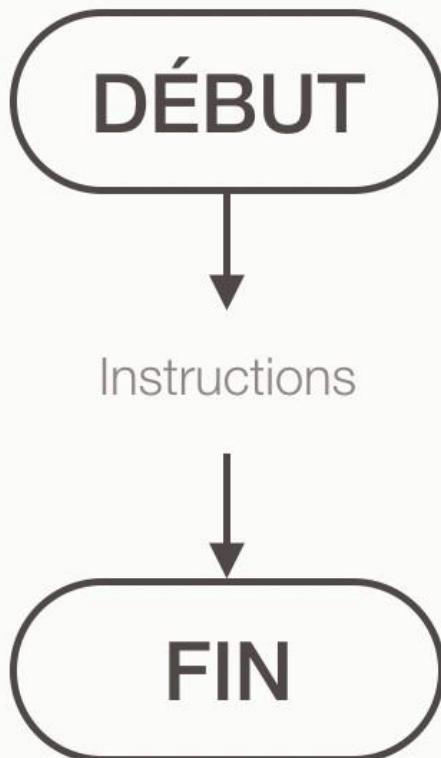
"Algorithme graphique"

Représentation graphique normalisée de l'**enchainement des opérations**.

Les opérations/instructions se trouvent entre les cartouches de **début** et de **fin**.

Quel que soit le chemin pris à partir du cartouche de début, l'algorithme doit permettre d'atteindre le cartouche de fin.

Les **flèches** indiquent la direction à prendre.

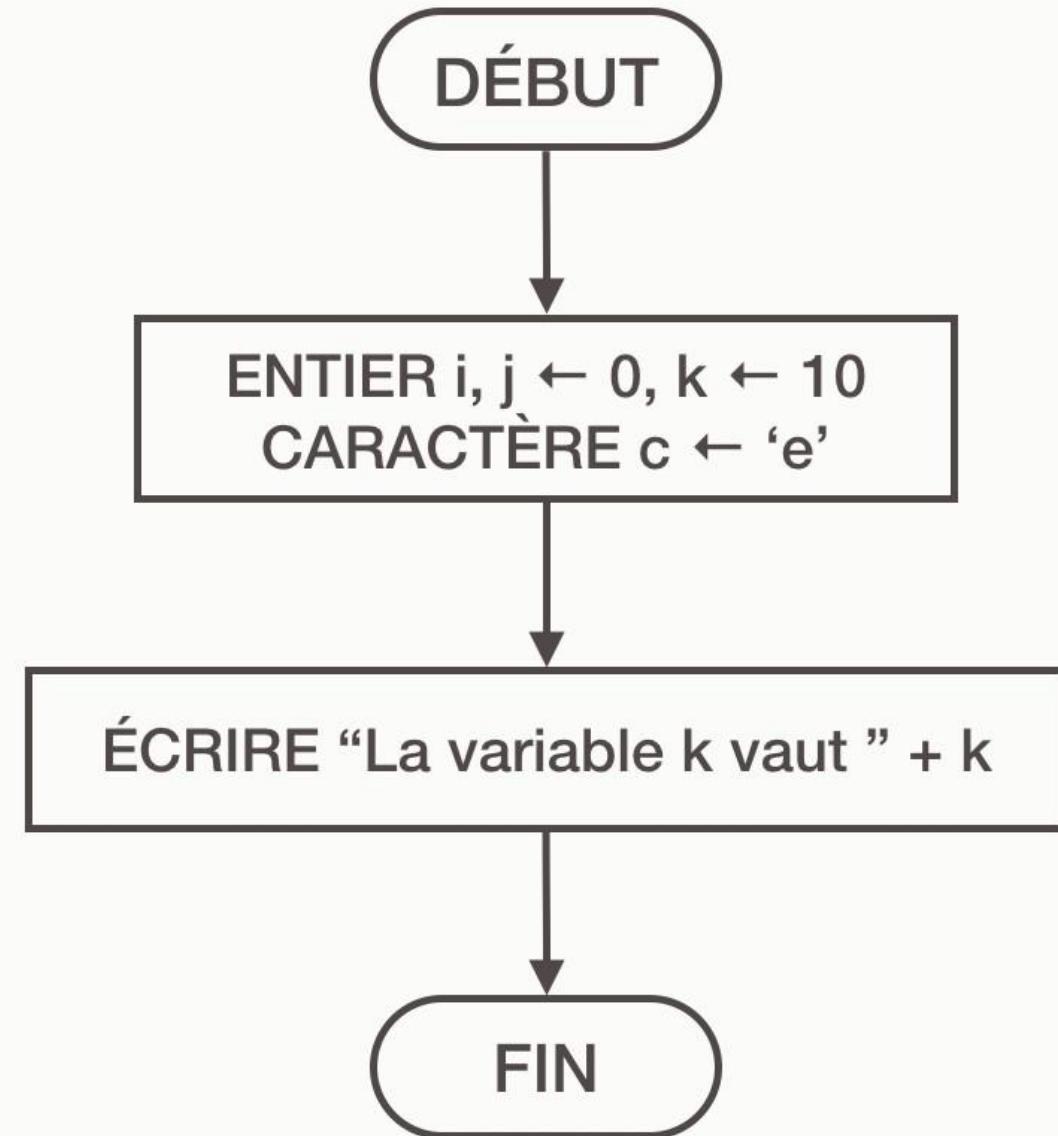


Logigramme

Les instructions

Les instructions se font dans un rectangle, entre le cartouche de début et le cartouche de fin.

👉 Une bonne méthode consiste à initialiser toutes les variables, on voit !
ici que **i** n'est pas initialisée. D'ailleurs, **i**, **j** et **c** sont inutilisées, il vaut



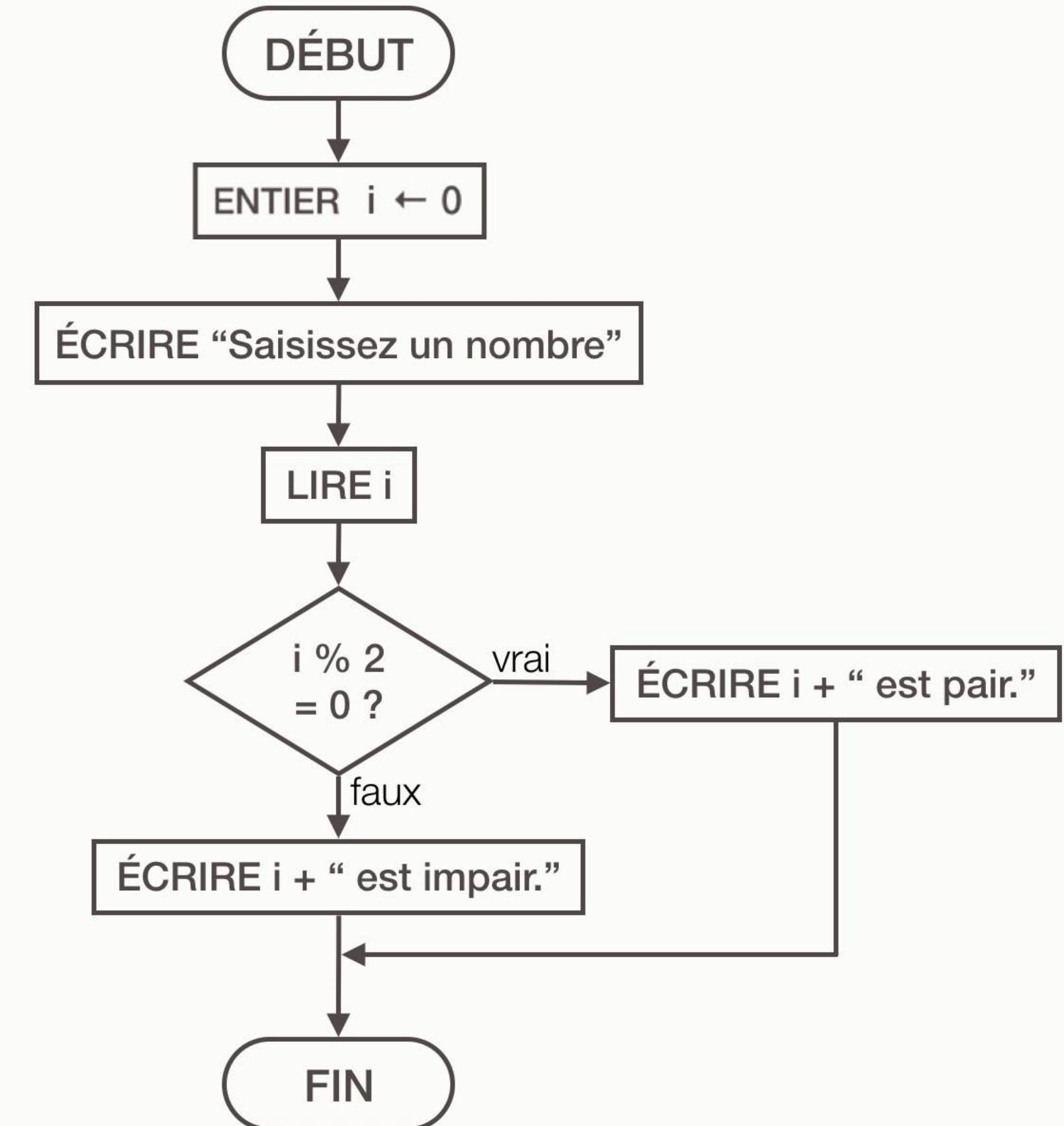
Logigramme

mieux éviter toute chose inutile ! On devrait donc supprimer la déclaration de **i**, **j** et **c**.

Les tests

Les tests se font dans un losange avec deux sorties :

- ◎ **vrai** : la condition est vérifiée.
- ◎ **faux** : la condition n'est pas vérifiée.



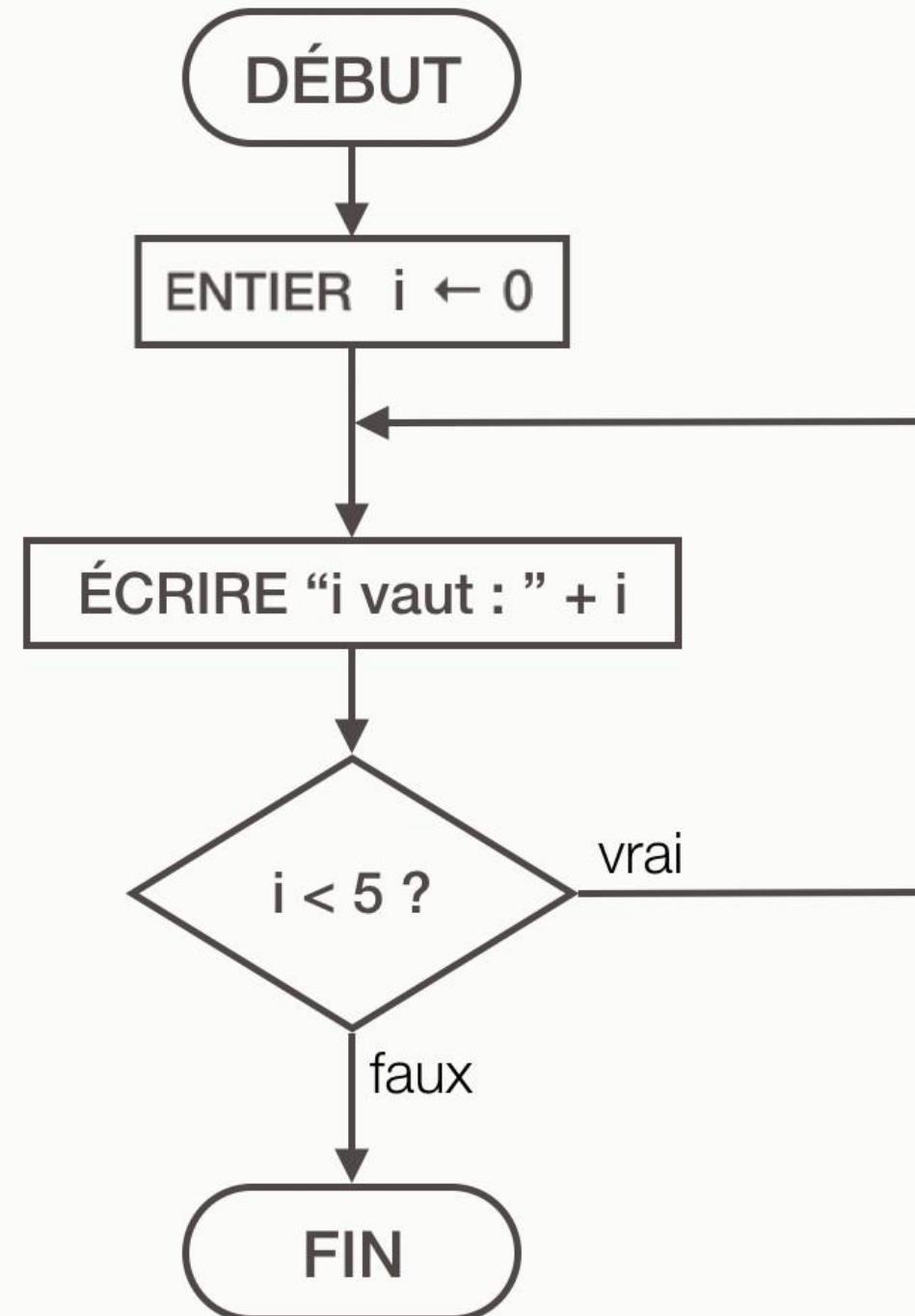
Logigramme

Les boucles

Les boucles sont semblables aux tests, à l'exception près qu'une des deux sorties remonte là d'où on vient.

Quelque chose ne va pas dans le logigramme ci-contre.

Qu'affiche-t-il ? Que faut-il changer ?

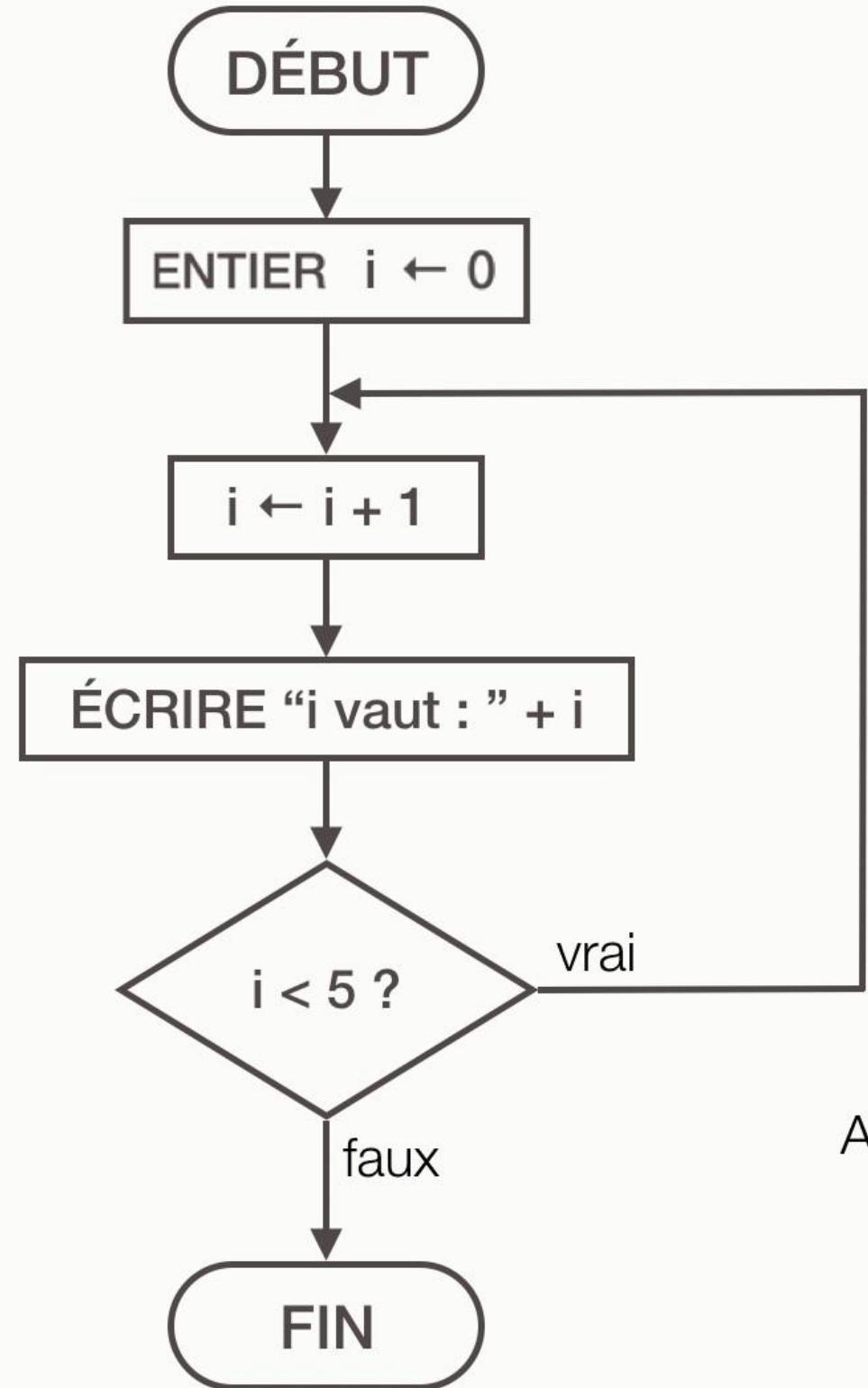


Logigramme

Les boucles - version 1

Les boucles sont semblables aux tests, à l'exception près qu'une des deux sorties remonte là d'où on vient.

⚠️ Attention à ne pas oublier d'**incrémenter** (= augmenter la valeur d'une variable suivant un incrément donné : **1** par défaut) l'**itérateur** (= variable utilisée dans une boucle comme indice de parcours).



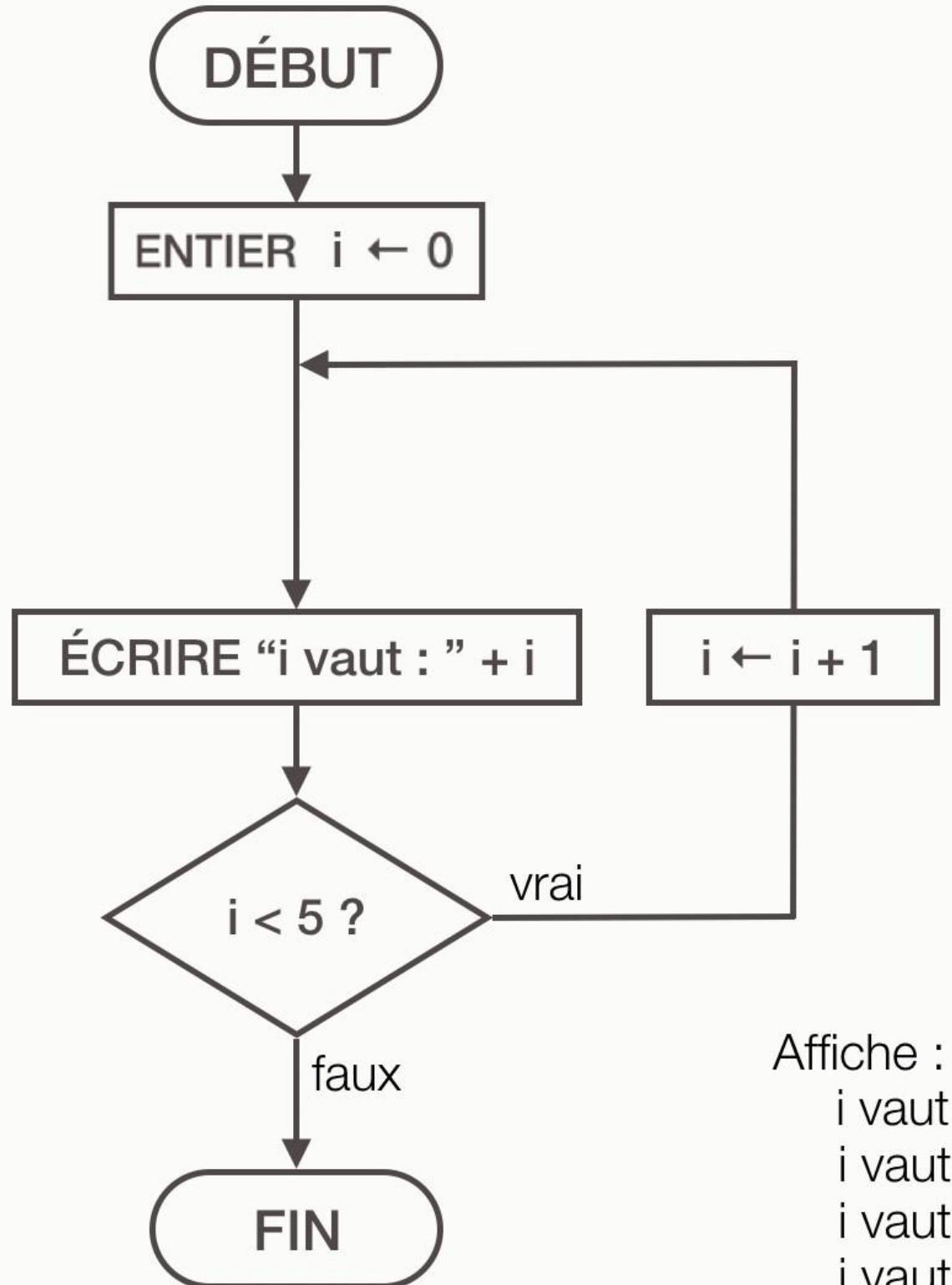
Affiche :
i vaut 1
i vaut 2
i vaut 3
i vaut 4
i vaut 5

Logigramme

Les boucles - version 2

Les boucles sont semblables aux tests, à l'exception près qu'une des deux sorties remonte là d'où on vient.

⚠️ Attention à ne pas oublier d'**incrémenter** (= augmenter la valeur d'une variable suivant un incrément donné : **1** par défaut) l'**itérateur** (= variable utilisée dans une boucle comme indice de parcours).

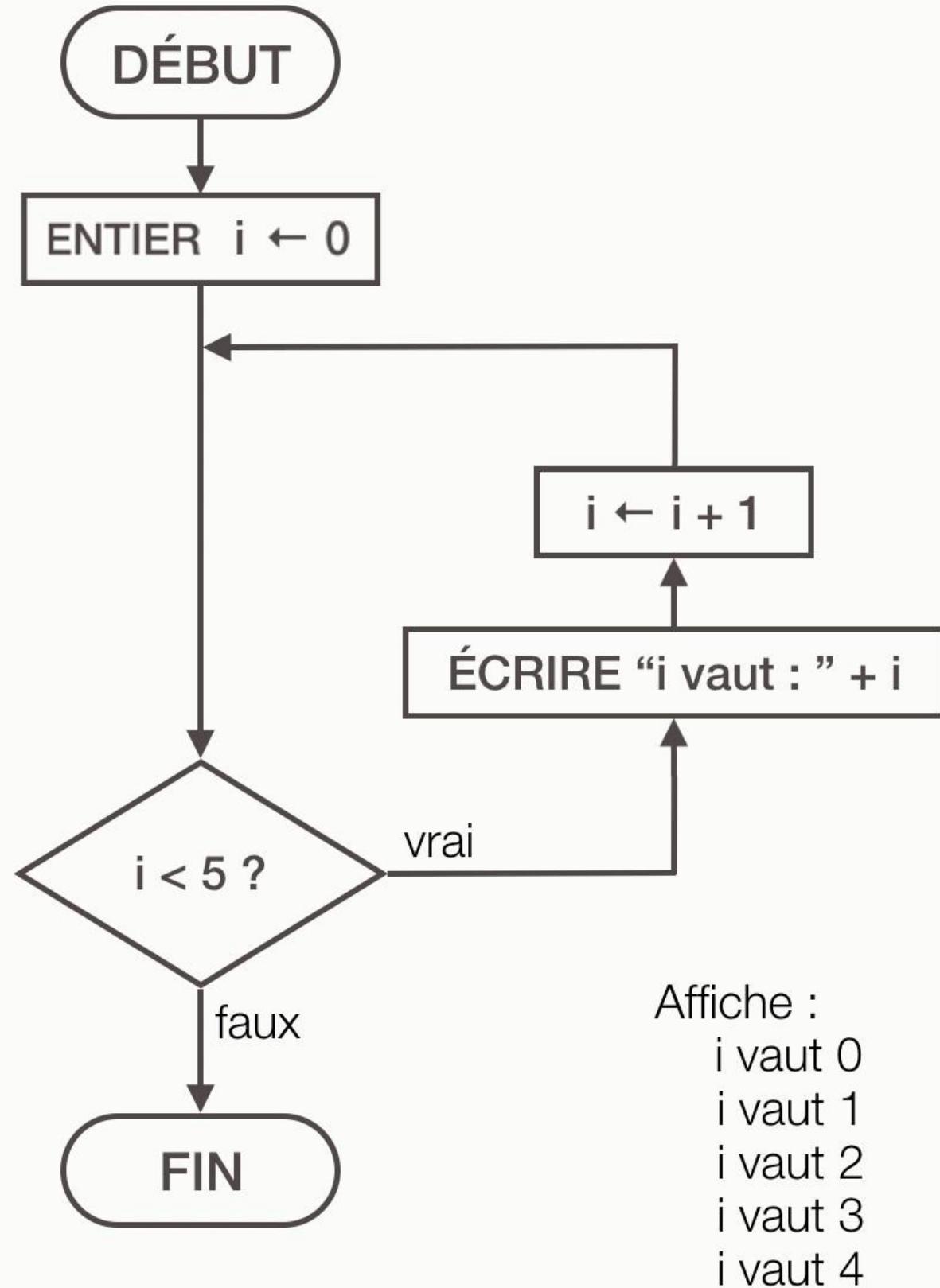


Logigramme

Les boucles - version 3

Les boucles sont semblables aux tests, à l'exception près qu'une des deux sorties remonte là d'où on vient.

⚠️ Attention à ne pas oublier d'**incrémenter** (= augmenter la valeur d'une variable suivant un incrément donné : **1** par défaut) l'**itérateur** (= variable utilisée dans une boucle comme indice de parcours).





Exercice/Exemple 1

Nombre positif, négatif ou nul ?

Donnez un algorithme ainsi que le logigramme correspondant permettant de répondre au cahier des charges suivant :

Nous souhaitons un programme qui demande à l'utilisateur un nombre au clavier, avant de lui indiquer si ce nombre est négatif, positif ou nul.



Exercice/Exemple 1

Nombre positif, négatif ou nul ?

Correction

DÉBUT

ENTIER $n \leftarrow 0$

ÉCRIRE "Saisissez un nombre"

LIRE n

SI $n = 0$ ALORS

 ÉCRIRE $n +$ " est nul."

SINON SI $n > 0$ ALORS

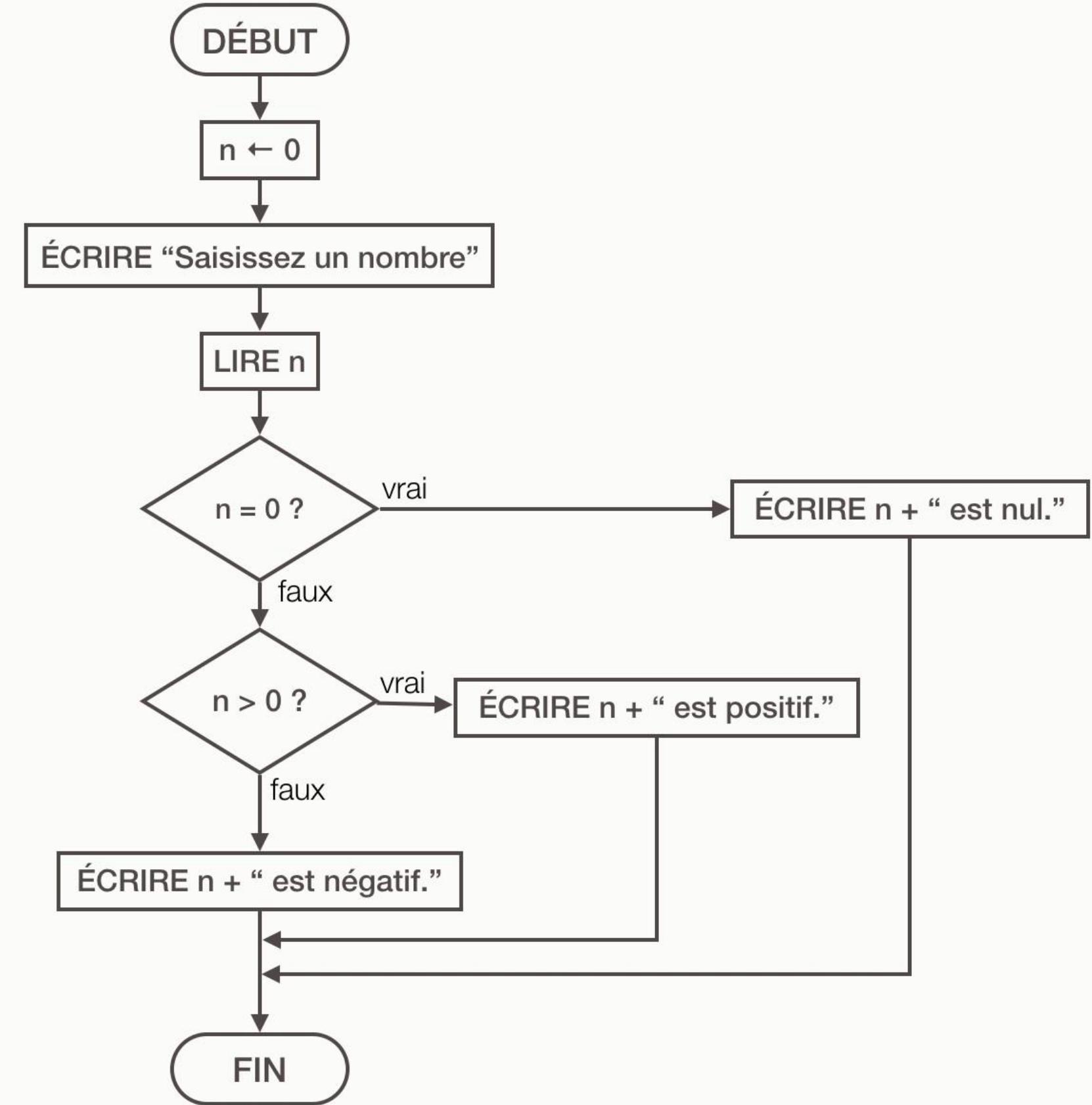
 ÉCRIRE $n +$ " est positif."

SINON

 ÉCRIRE $n +$ " est négatif."

FIN SI FIN

NB : d'autres corrections sont possibles.





Moyenne de nombres négatifs

Donnez un algorithme et son logigramme permettant à un utilisateur de saisir autant de valeurs négatives qu'il le souhaite, puis d'en afficher la moyenne.

Les valeurs positives ne devront pas être prises en compte.



Exercice/Exemple 2

Moyenne de nombres négatifs Correction

DEBUT

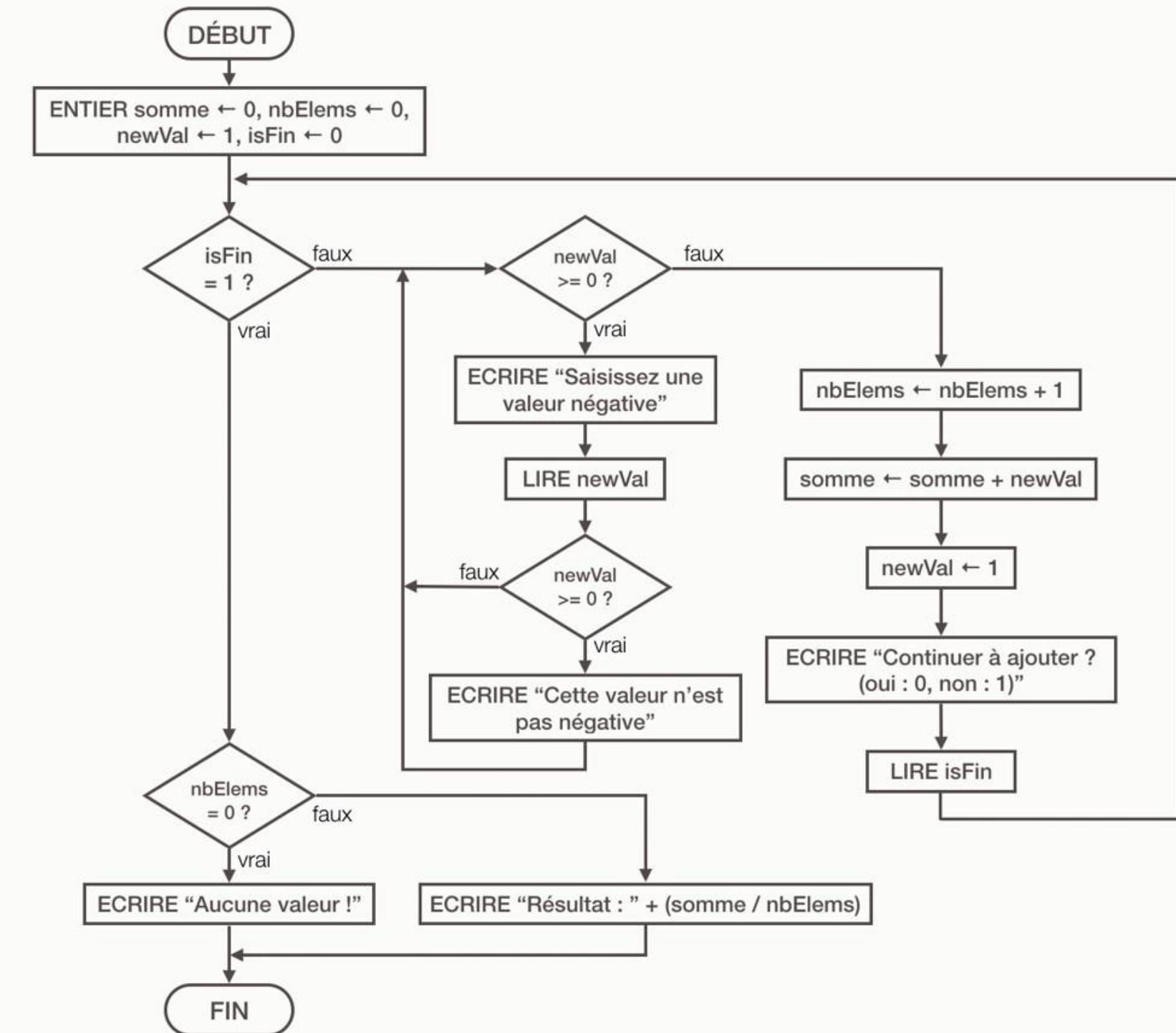
```
ENTIER somme ← 0, nombreElements ← 0, nouvelleValeur ← 1, isFin ← 0
TANT QUE isFin ≠ 1 FAIRE
    TANT QUE nouvelleValeur >= 0 FAIRE
        ECRIRE "Saisissez une valeur négative"
        LIRE nouvelleValeur
        SI nouvelleValeur > 0 FAIRE
            ECRIRE "Cette valeur n'est pas négative !"
        FIN SI
    FIN TANT QUE
    nombreElements ← nombreElements + 1
    somme ← somme + nouvelleValeur
    nouvelleValeur ← 1
    ECRIRE "Voulez-vous ajouter un autre nombre négatif à la moyenne ? (oui: 0, non: 1)"
    LIRE isFin
FIN TANT QUE
SI nombreElements ≠ 0 FAIRE
    ECRIRE "Résultat : " + (somme / nombreElements)
SINON
    ECRIRE "Vous n'avez saisi aucune valeur !"
FIN SI
FIN
```

Moyenne de nombres négatifs

Correction

Rappel de l'algorithme (slide précédente) :

```
DEBUT
    ENTIER somme ← 0, nombreElements ← 0, nouvelleValeur ← 1, isFin ← 0
    TANT QUE isFin ≠ 1 FAIRE
        TANT QUE nouvelleValeur >= 0 FAIRE
            ECRIRE "Saisissez une valeur négative"
            LIRE nouvelleValeur
            SI nouvelleValeur > 0 FAIRE
                ECRIRE "Cette valeur n'est pas négative !"
            FIN SI
        FIN TANT QUE
        nombreElements ← nombreElements + 1
        somme ← somme + nouvelleValeur
        nouvelleValeur ← 1
        ECRIRE "Voulez-vous ajouter un autre nombre négatif à la moyenne ? (oui: 0, non: 1)"
        LIRE isFin
    FIN TANT QUE
    SI nombreElements ≠ 0 FAIRE
        ECRIRE "Résultat : " + (somme / nombreElements)
    SINON
        ECRIRE "Vous n'avez saisi aucune valeur !"
    FIN SI
FIN
```



Mon algorithme est-il correct ?

Il est essentiel de vérifier que chaque algorithme conçu est correct

- Vérification de l'adéquation du résultat avec le cahier des charges.

- **Jeux d'essais** : tester dans tous les cas de figure possibles

Par exemple, dans le corrigé de l'exercice 1 précédent, il faut vérifier le comportement de l'algorithme avec un nombre négatif, un nombre positif, et 0.

- **Preuves de terminaison** : vérifier que l'algorithme se termine toujours (pas de boucle infinie), avec un résultat valide.

- **Traces d'exécution**

NB : Ces étapes sont idéalement effectuées avant l'implémentation, mais peuvent aussi être faites sur l'algorithme une fois implémenté dans un langage.

Une fois un algorithme implémenté, d'autres familles de tests interviennent.

NB : Bien que souvent négligés, les tests sont très importants car ils garantissent le bon fonctionnement de l'application tout au long de sa vie. En effet, il n'est pas rare qu'un ajout de fonctionnalité ou une modification du code existant "casse" une autre fonctionnalité. Par exemple, la modification d'un module peut engendrer un bug qui authentifie tous les utilisateurs, même s'ils fournissent un mauvais mot de passe. Dans ce cas, seuls les tests permettent de détecter cet **effet de bord**, à moins qu'un employé ou un utilisateur bien intentionné se trompe de mot de passe et détecte ce bug.

Les tests unitaires

Généralement écrits avant les portions de code qu'ils sont censés tester, les tests unitaires **vérifient le bon fonctionnement d'une partie précise d'un logiciel** ou d'un module d'un programme.

Exemple : vérifie que le module d'authentification des utilisateurs revoie bien un échec si le mot de passe fourni est faux.

Les tests d'intégration

Les tests d'intégration vérifient le bon fonctionnement de l'application une fois tous les modules — *déjà validés par les tests unitaires* — assemblés.

Les tests de validation

Les tests de validation visent à valider le fonctionnement global de l'application :

- Respect du cahier des charges et des exigences du client
- Interface utilisable et ergonomique
- Vérification des performances et de la robustesse

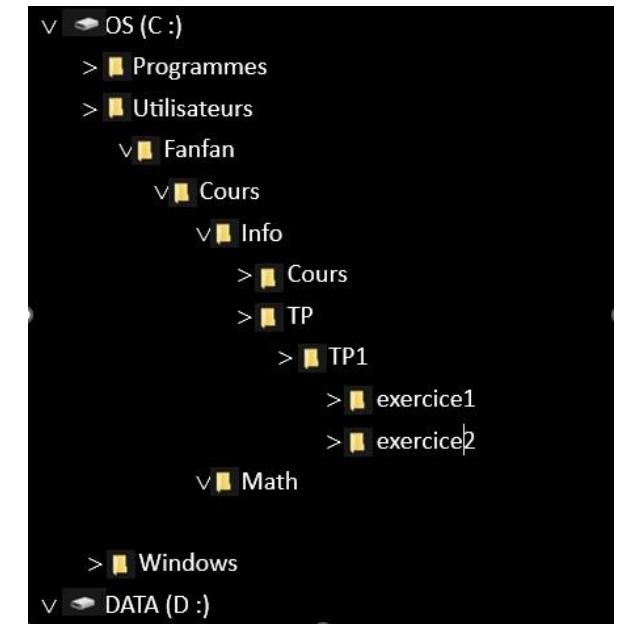
Comment l'application réagira si le nombre d'utilisateurs augmente considérablement ?

Espace de travail : dossiers, fichiers

- . Toutes les données stockées sur un support permanent [disque dur, CD-ROM, mémoire flash (clé USB)...] sont écrites dans des fichiers.
- . Un fichier porte un nom qui comporte souvent une extension (.txt, .doc, .jpg ...) indiquant son format.
- . Le format d'un fichier est la manière dont les données y sont codées en binaire et organisées. Il existe des centaines de formats de fichiers différents. Pour accéder au contenu d'un fichier, il faut

utiliser un programme ou un logiciel spécifique qui dépend de son format.

Les fichiers sont placés dans des emplacements appelés répertoires (dossiers). [En fait, un dossier est un fichier qui contient une liste de descriptions de fichiers. Chaque fichier ou dossier est référencé par un autre dossier. On obtient donc une arborescence de fichiers.]



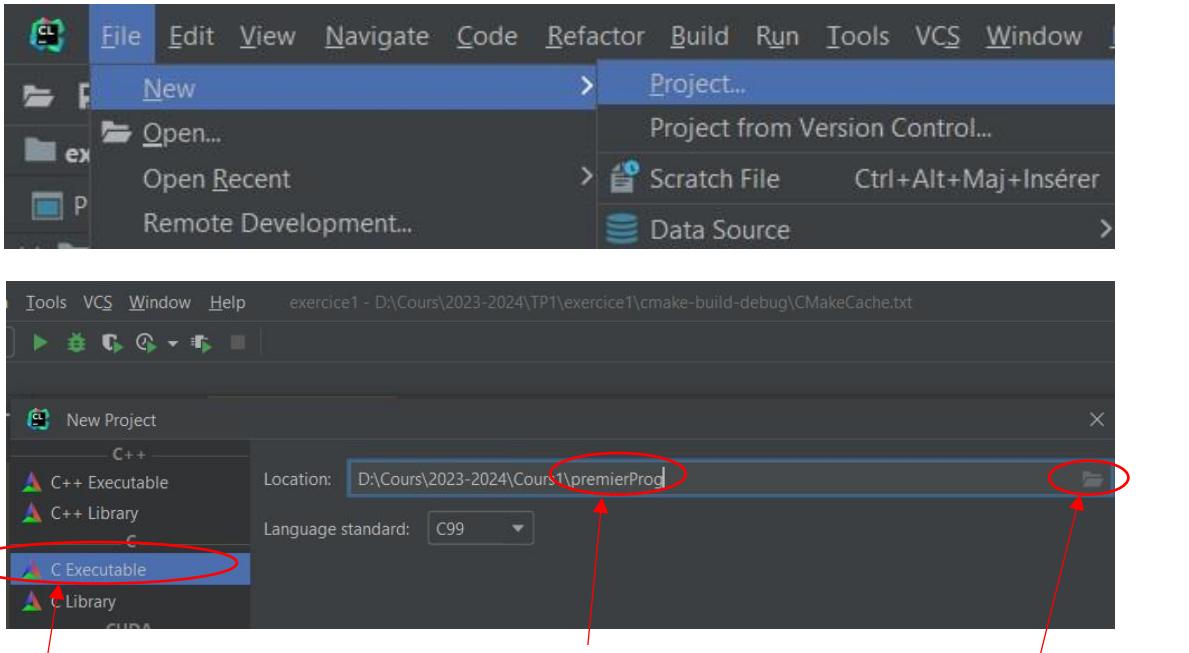
- Les programmes sources en langage C s'écrivent sous forme de texte en respectant la syntaxe et les mots-clés du langage. On peut utiliser n'importe quel éditeur de texte (bloc-notes, notepad++ ...).
- Les fichiers contenant du code source C portent l'extension .c
- Il faut ensuite compiler ces fichiers pour obtenir un programme binaire exécutable par l'ordinateur. On utilise pour cela un autre programme (un compilateur) par exemple.
- L'exécutable généré porte l'extension .exe

Différents logiciels Clion, CodeBlocks, CodeLite ...) offrent des environnements de développement intégré (IDE) permettant d'écrire, de compiler, de lancer et de débugger des programmes en langage C, tout en un ! Ils facilitent la gestion des projets multi-fichiers. CLion permet aussi du développement collaboratif avec partage de codes entre programmeurs avec l'outil GitHub qui sera utile pour les projets par équipe.

Le tutoriel d'installation et d'initiation de CLion se trouve dans le lien suivant :

<https://drive.google.com/file/d/1j1Mu27fEk0o0Q0TTJLwscJ3O9KDPeu18/view?usp=sharing>. De plus amples informations sur CLion vous seront fournies ultérieurement.

Dans les pages suivantes, vous avez des screenshot d'utilisation de CLion pour créer un nouveau projet en C et exécuter un programme.



3. Donner un nom au projet

1. Projet C Executable

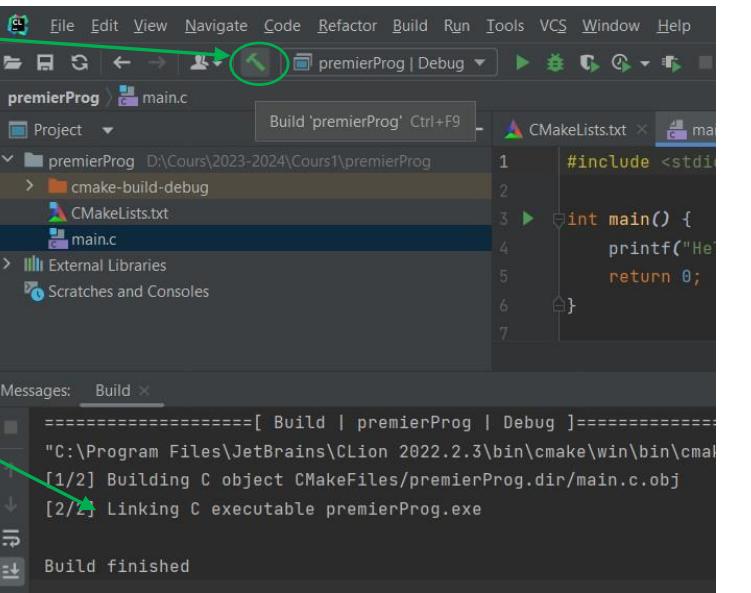
- Un nouveau dossier portant le nom du projet est automatiquement créé. Voici son contenu :

Nom	Mod
.idea	17/0
cmake-build-debug	17/0
CMakeLists.txt	17/0
main.c	17/0

```
#include <stdio.h>
int main() {
    printf("Hello, World!\n");
    return 0;
}
```

- **Le marteau vert permet de compiler le programme**

Résultats de la compilation. S'il y a des erreurs, elles seront indiquées ici.



```
#include <stdio.h>
int main() {
    printf("Hello World");
    return 0;
}
```

Messages: Build x

```
[ 1/2 ] Building C object CMakeFiles/premierProg.dir/main.c.o
[ 2/2 ] Linking C executable premierProg.exe
Build finished
```

La flèche verte permet de lancer le programme (si la compilation a réussi !)

Affichages lors de l'exécution.

[Ce programme ne fait rien d'autre que d'afficher « Hello, World ! »]

The screenshot shows a C IDE interface with a project named 'premierProg'. The code editor displays the 'main.c' file with the following content:

```
#include <stdio.h>
int main() {
    printf("Hello, World!");
    return 0;
}
```

The 'Run' toolbar at the top has several icons. A green arrow points from the text 'La flèche verte permet de lancer le programme' to the green play button icon in the toolbar. Another green arrow points from the text 'Affichages lors de l'exécution.' to the output window below. The output window shows the results of running the program: 'Hello, World!' followed by 'Process finished with exit code 0'.

Utilisation de chatGPT



L'utilisation d'outil comme chatGPT ou équivalent (ex : Copilot), se limitant à générer du code sans réflexion sérieuse de votre part, sera tracée et très sévèrement pénalisée par 0 et un avertissement sans négociation possible.

Il n'est pas interdit d'utiliser chatGPT car son utilisation peut vous être utile et parfois nécessaire mais à condition de l'utiliser de manière intelligente sans générer bêtement du code qui ne vous serait d'aucune utilité pour acquérir de réelles compétences de logique informatique et donc de compétences en programmation. A la place, nous vous proposons de lui demander des conseils utiles pour vous mieux vous guider dans la logique de tel ou tel problème plutôt que de vous donner sa propre solution souvent erronée.