INFORMATIQUE · 4TT · ALGORITHMIE

SYNTHÈSE GÉNÉRALE

Tout le cours d'algorithmie

L'essentiel des 7 chapitres et le lexique complet



LES CHAPITRES

1. Les variables et l'affectation
2. La lecture et l'écriture
3. Les conditions : SI, SINON, SINONSI
4. La priorité des opérateurs
5. Les boucles : TANTQUE et POUR
6. Du pseudocode au C# : la conversion
7. Les fonctions en C#

CHAPITRE 1

Les variables et l'affectation

1. Un **algorithme** est une suite d'instructions ordonnées qui mène à un résultat. On l'écrit en **pseudocode** avant de le traduire dans un langage comme le C#.
2. Quatre familles d'instructions suffisent : **affectation**, **lecture / écriture**, **tests** et **boucles**.
3. Une **variable** est une boîte étiquetée en mémoire : un **nom** (identifiant), un **type** et une seule valeur à la fois.
4. Un identifiant contient des lettres sans accent, des chiffres et `_`, sans espace, et ne commence pas par un chiffre.
5. Les quatre types : `entier`, `numerique`, `chaine de caracteres` (entre guillemets) et `booleen` (`true` ou `false`).
6. On **déclare** avant `DEBUT` (`variable age en entier`), puis on **affecte** avec `<-` : seule la variable de gauche change.
7. Les instructions s'exécutent **de haut en bas** ; un **tableau de trace** permet de les suivre pas à pas.

CHAPITRE 2

La lecture et l'écriture

1. **Écrire**, c'est afficher à l'écran ; **lire**, c'est recevoir une valeur tapée au clavier. On se place toujours du côté du programme.
2. `ECRIRE "texte"` affiche le texte ; `ECRIRE variable` affiche sa **valeur** ; chaque ECRIRE passe à la ligne.
3. La **concaténation** `&` colle textes et variables en une seule chaîne. Les espaces se mettent dans les guillemets. Jamais de virgule.

4. `LIRE variable` met le programme en pause et range la valeur tapée dans la variable, qui doit être déclarée avec le bon type.

5. Un `ECRIRE` qui pose la question vient toujours **avant** le `LIRE`.

6. On calcule avec `+`, `-`, `*`, `/` et `mod`, le **reste** de la division entière.

CHAPITRE 3

Les conditions : SI, SINON, SINONSI

1. Un **test** permet de choisir : `SI condition ALORS … FINSI` exécute les instructions seulement si la condition est vraie.

2. Avec `SINON`, on exécute un bloc ou l'autre, jamais les deux, puis on reprend après le `FINSI`.

3. Une **condition** est une comparaison : une valeur, un opérateur (`=`, `<>`, `<`, `>`, `<=`, `>=`) et une valeur du même type.

4. Pour des cas qui s'excluent, on enchaîne `SI … SINONSI … SINON … FINSI` : le premier test vrai l'emporte.

5. `20 < age < 30` est faux : on écrit `age > 20 ET age < 30`.

6. `ET` : les deux vraies. `OU` : au moins une. `XOR` : une seule. `NON(…)` inverse.

7. Un **booléen** peut servir directement de condition ; un test peut être **imbriqué** dans un autre, avec son propre `FINSI`.

CHAPITRE 4

La priorité des opérateurs

1. La **priorité** est l'ordre dans lequel la machine évalue une expression ; les **parenthèses** imposent un autre ordre.
2. Dans les calculs, `*`, `/` et `mod` passent avant `+` et `-` ; les calculs passent avant les comparaisons.
3. Dans une condition : **( )**, puis **NON**, puis les **comparaisons**, puis **ET**, et enfin **OU**.
4. Le **ET passe avant le OU** : `A OU B ET C` se lit `A OU (B ET C)`. Mets des parenthèses autour du OU.
5. `NON` s'écrit devant des parenthèses : `NON(age > 18)`.
6. **De Morgan** : `NON(A OU B)` = `NON(A) ET NON(B)` ; `NON(A ET B)` = `NON(A) OU NON(B)`.
7. Pour une règle complexe : découpe, traduis chaque morceau, relie, groupe, puis **teste** des cas.

CHAPITRE 5

Les boucles : TANTQUE et POUR

1. Une **boucle** répète une série d'instructions ; c'est la quatrième famille d'instructions.
2. `TANTQUE condition … FINTANTQUE` répète **tant que** la condition est vraie ; elle est testée avant chaque tour.
3. La variable testée doit avoir une valeur **avant** la boucle, et changer **dans** la boucle.
4. Le **contrôle de saisie** redemande tant que la réponse n'est pas valable.

5. Un **compteur** augmente de 1 (`i <- i + 1`) ; un **accumulateur** additionne (`total <- total + note`), en partant de `0`.

6. Pièges : la **boucle infinie** (rien ne change) et la boucle qui **ne démarre jamais** (fausse dès le départ).

7. `POUR i <- 1 à n … POUR SUIVANT … FINPOUR` quand le nombre de tours est connu ; sinon `TANTQUE`.

CHAPITRE 6

Du pseudocode au C# : la conversion

1. Un programme C# : `using System;`, `class Program`, `static void Main()` ; les **accolades** remplacent `DEBUT` et `FIN`.

2. Chaque instruction finit par `;`. Le C# distingue majuscules et minuscules.

3. Types : `entier` → `int`, `numerique` → `double`, `chaine de caracteres` → `string`, `booleen` → `bool` ; `char` pour un seul caractère entre apostrophes.

4. `<-` devient `=`, et le `=` qui compare devient `==` ; `<>` → `!=`, `&` → `+`, `ET` → `&&`, `OU` → `||`, `mod` → `%`.

5. Entre deux `int`, `/` fait une **division entière** : `7 / 2` vaut 3.

6. `ECRIRE` → `Console.WriteLine(…);` ; `LIRE` → `Console.ReadLine()`, qui renvoie un `string` : on le convertit avec `int.Parse(…)`.

7. `SI` → `if (…) { }`, `SINONSI` → `else if`, `TANTQUE` → `while`, `POUR` → `for (int i = 1; i <= n; i++)`.

CHAPITRE 7

Les fonctions en C#

1. Une **fonction** regroupe des instructions sous un nom ; on l'écrit une fois et on l'**appelle** autant de fois qu'on veut.
2. `Console.WriteLine()` et `Console.ReadLine()` sont des fonctions toutes faites ; `Main` est la fonction principale, le **point d'entrée**.
3. Une fonction a un **type de retour**, un **nom**, des **paramètres** (facultatifs) et un **corps** entre accolades.
4. En C#, chaque fonction du programme commence par `static`.
5. `return` renvoie la valeur et **termine** la fonction ; une fonction `void` ne retourne rien.
6. Les paramètres sont passés **par copie** : modifier un paramètre ne change pas la variable d'origine.
7. Pour récupérer un résultat : `int noteArrondie = Arrondir(noteBrute);`.

LEXIQUE COMPLET

Tous les mots du cours

Accumulateur ch. 5
Variable qui additionne des valeurs au fil des tours.

Affectation ch. 1
Ranger une valeur dans une variable avec la flèche `<-`.

Algorithme ch. 1
Suite d'instructions ordonnées qui mène à un résultat.

Appel ch. 7
Utiliser une fonction : son nom et ses valeurs entre parenthèses, `Additionner(5, 10)`.

Bloc ch. 6
Instructions entre accolades `{ }` ; une variable n'existe que dans son bloc.

Booléen ch. 1
Type qui ne vaut que `true` (vrai) ou `false` (faux).

Boucle ch. 5
Structure qui répète des instructions (`TANTQUE`, `POUR`).

Boucle infinie ch. 5
Boucle dont la condition reste toujours vraie : elle ne s'arrête jamais.

C# ch. 6
Langage de programmation de Microsoft ; VS Code y traduit ton pseudocode.

Cas limite ch. 4
Une valeur à la frontière d'une condition (18 pour « 18 ans et plus ») : à toujours tester.

Cast ch. 6
Conversion entre types numériques : `(int)3.99` donne 3.

Chaîne de caractères ch. 1
Du texte, toujours écrit entre guillemets.

Compteur ch. 5
Variable qui augmente de 1 à chaque tour.

Concaténation ch. 2
Coller textes et valeurs en une seule chaîne avec `&`.

Condition ch. 3
Comparaison qui vaut `true` ou `false`.

Condition composée ch. 3
Plusieurs conditions reliées par un opérateur logique.

Conjonction ch. 4
L'opérateur `ET` : vrai si les deux conditions sont vraies.

Contrôle de saisie ch. 5
Boucle qui redemande une valeur tant qu'elle n'est pas valable.

Déclaration ch. 1
Annoncer le nom et le type d'une variable, avant `DEBUT`.

Disjonction ch. 4
L'opérateur `OU` : vrai si au moins une condition est vraie.

Division entière ch. 6
Division entre deux int : seule la partie entière est gardée.

Écriture ch. 2
Le programme affiche une information à l'écran (`ECRIRE`).

Fonction préexistante ch. 7
Fonction déjà écrite dans les bibliothèques du C# : `Console.WriteLine` , `Math.Max` ...

Identifiant ch. 1
Le nom d'une variable.

Itération ch. 5
Un tour de boucle.

Loi de De Morgan ch. 4
Règle pour faire entrer un NON dans une parenthèse : OU devient ET, ET devient OU.

Modulo ch. 2
Le reste de la division entière (`mod`) : `17 mod 5` vaut 2.

Opérateur de comparaison ch. 3
`=` , `<>` , `<` , `>` , `<=` , `>=` .

Paramètre ch. 7
Valeur reçue par la fonction, déclarée entre parenthèses avec son type.

Passage par copie ch. 7
Le paramètre reçoit une copie de la valeur : l'original ne change pas.

Priorité des opérateurs ch. 4
L'ordre d'évaluation des morceaux d'une expression.

Quotient ch. 2
Le résultat de la division entière : 17 / 5 donne 3.

Écran d'exemple ch. 2
Ce qu'un programme doit afficher, ligne par ligne, avec les valeurs tapées.

Fonction ch. 7
Instructions regroupées sous un nom, qui réalisent une tâche précise.

Groupement ch. 4
Utiliser des parenthèses pour imposer l'ordre d'évaluation.

Initialisation ch. 6
Déclarer et affecter en une ligne : `int note = 14;` .

Lecture ch. 2
Le programme attend une valeur tapée au clavier et la range dans une variable (`LIRE`).

Main ch. 6
La fonction lancée en premier : le point d'entrée du programme C#.

Négation ch. 4
L'opérateur `NON` , qui inverse true et false.

Opérateur logique ch. 3
`ET` , `OU` , `XOR` , `NON` : ils relient ou inversent des conditions.

Parse ch. 6
Conversion d'un texte en nombre : `int.Parse("42")` .

Portée ch. 6
La partie du code où une variable existe.

Pseudocode ch. 1
Langage d'apprentissage en français, traduit en C# par l'extension de VS Code.

return ch. 7
Renvoie une valeur et termine la fonction.

Sensible à la casse ch. 6
Majuscules et minuscules comptent : `Age` et `age` sont différents.

SINONSI ch. 3
Teste un nouveau cas quand tous les précédents étaient faux.

static ch. 7
Mot-clé placé devant chaque fonction du programme.

Tableau de trace ch. 1
Tableau qui suit la valeur des variables instruction après instruction.

Test ch. 3
Instruction qui choisit entre plusieurs chemins selon une condition (`SI`).

Test imbriqué ch. 3
Un test placé à l'intérieur d'un autre test.

Type ch. 1
La sorte de valeur qu'une variable peut contenir (entier, numerique, texte, booléen).

Type de retour ch. 7
Le type de la valeur renvoyée par la fonction (`int` , `string` ... ou `void`).

Valeur sentinelle ch. 5
Valeur spéciale qui arrête la saisie, comme -1.

Variable ch. 1
Emplacement de la mémoire qui garde une valeur, avec un nom et un type.

void ch. 7
« Vide » : la fonction ne retourne rien.

XOR ch. 3
Le « ou exclusif » : vrai si une seule des deux conditions est vraie.