Algorithmique et structure de données

Programme :

Chapitre 1 : Introduction
Chapitre 2 : Algorithme séquentiel simple
Chapitre 3 : Les structures conditionnelles.
Chapitre 4 : Les boucles.
Chapitre 5 : Les tableaux et les chaines de caractères
Chapitre 6 : les types personnalisés

Chapitre I : Introduction (Informatique & Algorithmique)

1. Introduction

1.1. L'informatique

Le mot informatique a été créé en 1962 par Philippe Dreyfus. Il s'agit d'une contraction des deux mots « information » et « automatique » : ***INFOR****mation auto****MATIQUE.***
L'informatique est la science du traitement automatique de l'information grâce à l'ordinateur, C-à-d automatiser l'information que nous manipulons. Elle a pour objet d'élaborer et de formuler l'ensemble de commandes, d'ordres ou d'instructions permettant de commander l'ordinateur et de l'orienter lors du traitement.

1.2. Brève histoire de l'informatique

L'informatique est apparue au milieu du 20ème siècle, elle a connu une évolution extrêmement rapide. L'informatique est présente dans tous les domaines de notre vie quotidienne : industrie, gestion, calculs scientifiques et techniques, enseignement, télécommunication, jeux, etc. L'histoire de l'informatique résulte de la conjonction entre des découvertes scientifiques de plusieurs disciplines techniques et sociales. La progression de l'automatique de l'électronique et des techniques de calculs a contribué fortement à la naissance et l'évolution rapide de l'informatique

1.2.1 Evolution de l'informatique et des ordinateurs

– En 1643, BLAISE PASCAL invente la "PASCALINE" : machine mécanique capable de réaliser des additions et soustractions.
– En 1812, "CHARLES BABBAGE" a conçu une machine mécanique pouvant effectuer des calculs numériques compliqués (c'est une machine à base de cartes perforées). En 1860, il définit les grands principes des calculateurs électroniques.
– En 1885, "HERMANN HOLLERITH" (inventeur des cartes perforées) construit la première machine à cartes perforées et qui a servit dans l'opération de recensement de la population d'Amérique en 1890.
En 1946 le premier calculateur électronique est apparu. Il est baptisé "ENIAC" (Electronic Numerical Integrator and Computer). Il est construit sur le principe du binaire (0 et 1) : (le courant passe ou ne passe pas). C'est une machine électronique composées de :
– ~ 6000 relais (commutateurs) mécaniques
– ~ 1800 tubes électroniques
– ~ 70000 résistances
Et pèse environ 30 tonnes. Il est capable de réaliser ~5000 additions/seconde et ~3000 multiplications/seconde. C'est le premier ordinateur qui utilise le principe de "Programme enregistré" et constitue la première génération des ordinateurs.

Première Génération d'ordinateurs (1946 – 1957) :

Ceux sont les ordinateurs construits sur la base de "tubes électroniques" comme :
– ENIAC (1946)
– IBM 701 (1052)
– Etc …

Deuxième Génération d'ordinateurs (1958 – 1963) :

Ils sont apparus après l'invention du "Transistor" composant électronique capable de réguler le courant. Ces ordinateurs sont 100 plus rapides que ceux de la 1ère génération et consomment moins d'énergie électrique et sont moins volumineux (occupent moins d'espaces).
Exemples :

– PDP I (1ère ordinateur de 2ème génération (1960))
– IBM 7030 (1961)

Troisième Génération d'ordinateurs (1964-1971) :

Après l'apparition du "Circuit Intégré", ces types d'ordinateurs utilisent les transistors et les circuits intégrés. Exemple de ces ordinateurs :
– IBM 360 (1964)
– DEC PDP8 (1964)

Quatrième Génération d'ordinateurs :

Après l'apparition du "microprocesseur". Le premier microprocesseur est fabriqué par la société INTEL en 1971. Les ordinateurs (microordinateur) qui sont construits autour de microprocesseur constituent les ordinateurs de la 4ème génération.
Exemple de microordinateurs:
– MICRAL 8008 (1973)
– ALTAIR 8008 (1973)
– APPLE1 (1976)
– IBM PC (1981)
– PENTUIM etc. ...

Actuellement l'évolution des ordinateurs tend à exploiter le laser, les fibres optiques, la biochimie ou ordinateur moléculaire.

2. Introduction à l'algorithmique

Le mot « algorithme » vient de la transcription latinisée d'Al-Kwharizmi, nom d'un célèbre mathématicien arabe, et du mot grec arithmos qui signifie « nombre ».

2.2 Exemples Introductifs :

Exemple 1 : Je suis dans la rue, et je veux rentrer à la maison qui se trouve dans un immeuble équipé d'un ascenseur. Proposer un algorithme pour prendre l'ascenseur ?

- **Prendre l'ascenseur**
    1. Appuyer sur le bouton d'appel
    2. Ouvrir la porte de l'ascenseur
    3. Entrer dans l'ascenseur
    4. Attendre la fermeture de la porte
    5. Appuyer sur le bouton de l'étage
    6. Descendre de l'ascenseur

Exemple 2 :

- **Additionner 2 nombres à l'aide d'une calculatrice :**
    1. Allumer la Calculatrice.
    2. Taper le 1er nombre.
    3. Appuyer sur la touche (+)
    4. Taper le 2eme nombre.
    5. Appuyer sur la touche (=)

2.3 Définitions :

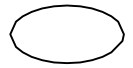
Algorithme : un algorithme est une suite d'actions (opérations) à exécuter, pour l'obtention d'une solution (résultat) à un problème posé. Donc, plus précisément, un algorithme est une description d'un traitement destiné à être réalisé sur un ordinateur grâce à un langage de programmation pour accomplir une tâche donnée.
Un algorithme est appelé aussi **« pseudo-code ».**

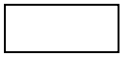
L'algorithmique : désigne la discipline qui étudie les algorithmes et leurs applications en informatique

Action : est une étape d'algorithme, c'est un évènement de durée déterminée.

- Dans les exemples 1 et 2, chaque étape est une action.
- Il est important de noter que : la séquence (l'ordre) d'actions doit être respectée (1, 2, 3, 4, 5).

Organigramme : est une représentation graphique de l'enchaînement d'une suite d'actions. Un organigramme utilise des symboles graphiques :

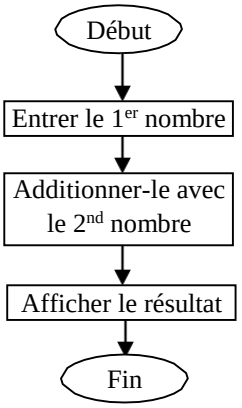

Le début et la fin d'un traitement


Pour la majorité des actions


Test (condition)


Sens des actions

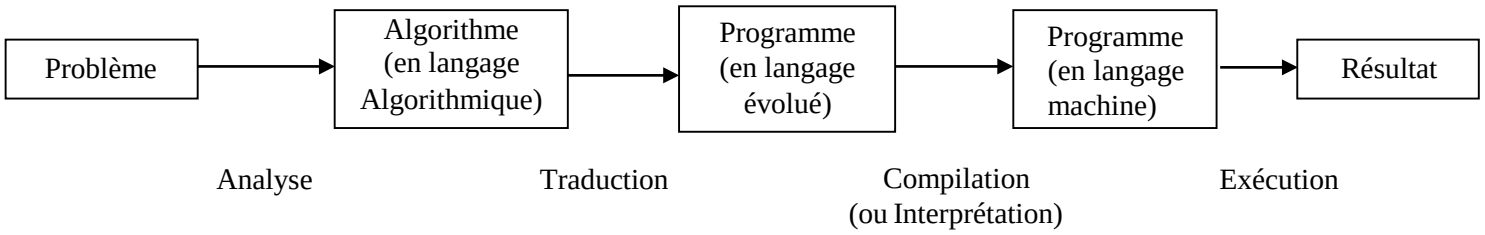
La somme de 2 nombres à l'aide d'une calculatrice dans l'exemple 2 peut-être schématisée comme suit :



2.4 Propriétés d'un algorithme :

. Le concepteur d'un algorithme doit prévoir tous les cas possibles d'exécution.
. Un algorithme doit être toujours fini (nombre d'actions fini)
. La solution d'un même problème, peut être décrite avec plusieurs algorithmes (façons).
. Un algorithme est plus efficace qu'un autre, s'il est plus clair et plus rapide (moins d'actions)
. Pour la mise en pratique d'un algorithme sur l'ordinateur, il faut passer par un langage de programmation ou programme.

Du problème au résultat :

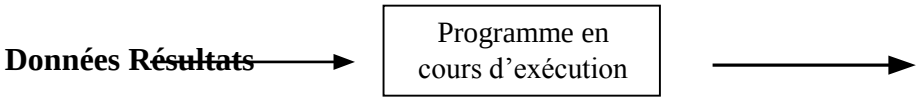


- **Problème** :
  exemples…

- **Analyse** : phase de réflexion qui permet de :
  - Comprendre le PB
  - Dégager les Données (entrées)
  - Dégager les résultats (sorties)
  - Formaliser la solution trouvée par un algorithme
- **Algorithme :** Suite organisée d'actions pour l'obtention d'une solution à un pb posé
- **Traduction :** pour son exécution sur ordinateur, un algorithme doit être transformé en un programme exprimé par un langage de programmation tel que C, Java,…
- **Programme :** un programme informatique est une succession (suite) d'instructions (actions) exécutables par l'ordinateur. Ces instructions écrites dans un langage de programmation et traduisant un algorithme. Les programmes sont traduits en binaires (0 et 1) par un compilateur, pour qu'ils soient compréhensibles par la machine. Exemple de

langage de programmation (langage évolué) : Pascal, C, C++, Basic, …

- **Compilateur** : est un programme qui permet de traduire un texte écrit dans un langage de programmation en langage machine (0,1).
- **Exécution :** l'exécution se fait en séquence (La 1ere instruction, puis la 2° instruction,….)
  Pendant l'exécution :



Exemple 3 :

Pb : Calculer la moyenne de 2 nombres entiers X et Y.

Analyse : Pour calculer la moyenne de 2 nombres X et Y, il faut additionner ces deux nombres, puis diviser le résultat par 2.

Données : X , Y. Résultat : Moyenne.

Algorithme

1. Lecture de X et Y.
2. Calculer : Somme = X+Y.
3. Calculer : Moyenne = Somme/2.
4. Afficher le résultat (Moyenne).

Chapitre II : Algorithme séquentiel simple

1. Notion de langage et langage algorithmique

Langage Algorithmique : Tout algorithme est exprimé dans un langage naturel appelé **Langage Algorithmique (LA) ou pseudo-langage,** il décrit de manière structurée, claire et complète, les objets manipulés par l'algorithme ainsi que l'ensemble des instructions à exécuter sur ces objets pour obtenir des résultats.

2. Parties d'un algorithme

Un algorithme se compose de deux grandes parties :

• les informations dont on a besoin au départ (Entête) ;

• la succession d'instructions à appliquer (Corps)

2.1 Structure générale d'un Algorithme : un algorithme doit avoir un nom, des déclarations et un groupe d'actions (corps de l'algorithme). La forme générale d'un algorithme est la suivante :

```
Algorithme Nom de l'algorithme

Déclaration
        Constante < définition de constantes>
        Type      < définition de types>
        Variable <définition de variables>
Début
        Action 1
        Action 2
                ……. <Partie Instructions> (Corps)
        Action n

Fin
```

L'algorithme est composé de trois parties :

Entête : qui spécifie le nom de l'algorithme par exemple : **Algorithme** Somme ;

Déclaration : contient la déclaration de différents objets (constants, variables, ..) utilisées avec leurs types, ainsi que les modules (fonctions ou procédures) utilisés.

Corps : Le corps d'un algorithme consiste en une suite d'opérations élémentaires (actions ou instructions), peuvent faisant appel à des fonctions ou procédures. Ces différentes opérations sont délimitées par les mots clefs **Début** et **Fin**.(nous découvrirons les fonctions et les procédures dans la suite du cours)

3. Les données : variables et constantes

Notion d'objet(ou Identificateur): Tous les objets qui rentrent en jeu pour l'exécution d'un algorithme doivent être déclarés, dans la partie déclaration, avant toute utilisation. Un objet peut être décrit par un **nom** (significatif), un **type**(type prédéfini), une **valeur** et une **nature** (variable, constante). Exemples :Somme, Moy, Calcul, X, Y,…

3.1 Notion de variable

Une Variable est un objet manipulé par un algorithme dont la valeur peut être modifiée durant l'exécution (valeur changeable) et qui peut être :

- une donnée d'entrée ;
- un résultat intermédiaire d'un calcul.
- le résultat final d'un calcul ;

Remarque :

1. Une fois qu'un type de données est associé à une variable le contenu de cette variable doit obligatoirement être du même type.

2. Si plusieurs variables sont de même type, nom-de-variable est remplacée par une suite de noms de variables séparés par des virgules.

Règle: La variable doit être déclarée avant d'être utilisée, elle doit être caractérisée par :

- ✓ Un nom (Identificateur)
- ✓ Un type qui indique l'ensemble des valeurs que peut prendre la variable (entier, réel, booléen, caractère, chaîne de caractères, …)
- ✓ Une valeur

Règles sur les identificateurs :

Le choix du nom d'une variable est soumis à quelques règles qui varient selon le langage, mais en général :

- Un nom doit commencer par une lettre alphabétique. Exemple : E1 (1E n'est pas valide)
- Doit être constitué uniquement de lettres, de chiffres et du soulignement (« _ ») (Éviter les caractères de ponctuation et les espaces). Exemples : SMI2008, SMI_2008. (SMP 2008, SMP-2008, SMP;2008 : sont non valides)
- Doit être différent des mots réservés du langage (par exemple en C: int, float, double, switch, case, for, main, return, …)
- La longueur du nom doit être inférieure à la taille maximale spécifiée par le langage utilisé (en c, le nom doit faire au plus 32 caractères).
- En c, Les majuscules sont distinguées des minuscules : a et A sont deux noms différents.

Conseil : pour la lisibilité du code choisir des noms significatifs qui décrivent les données manipulées.
Exemples : NoteEtudiant, Prix_TTC, Prix_HT

3.1.2 Déclaration d'une variable

Syntaxe : Variable Nom-de-variable : Type

Exemples : Variable Compteur : Entier
Lettre : Caractère
Rayon : Réel

3.2 Les constantes :

Une constante est un objet de nature constante dont la valeur est inchangeable pendant l'exécution d'un algorithme (programme). Elle peut être un nombre, un caractère, ou une chaine de caractères.

Déclaration des constantes :

Syntaxe : Constante Nom-de-constante = Valeur

Exemples : **Constante** PI = 3.141559
N = 10
Titre = 'Algo'

4. Type de données

Un type caractérise les valeurs que peut prendre une variable. Il définit également les opérations, généralement appelées opérateurs, qui pourront être appliquées sur les données de ce type.

Les types ont deux intérêts principaux :

– Permettre de vérifier automatiquement (par le compilateur) la cohérence de certaines opérations. Par exemple, une valeur définie comme entière ne pourra pas recevoir une valeur chaîne de caractères.

– Connaître la place nécessaire pour stocker la valeur de la variable. Ceci est géré par le compilateur du langage de programmation considéré et est en général transparent pour le programmeur. En algorithmique les types de données de base sont : l'entier, le réel, le booléen, le caractère et la chaine de caractère.

Le type entier : Désigne les nombres entiers (ex : 2, 7, -74,0,..)

Le type réel : Désigne les nombres réels (ex : 12, 7.39, -4,3,..)

Le type logique :Désigne les variables peuvent avoir des valeurs logique vrai ou faux

Le type caractère : Désigne l'ensemble des lettres alphabétiques majuscules et minuscules, les chiffres, les caractères spéciaux (' ?', '!','$', '*', ….) et le caractère espace. Un caractère est représenté entre quottes.

Le type chaine de caractère : Désigne l'ensemble des chaines que nous pouvons former par la composition des caractères. Une chaine est représenté, aussi, entre quottes. (ex : 'première', 'année','licence').

5. Opérations de base

Pour pouvoir comprendre et utiliser correctement les opérations de base en algorithmique, on doit tout d'abord aborder les notions d'opérateur, d'opérande et d'expression.

5.1. Opérateurs :

À chaque type est associé un ensemble d'opérations (ou opérateurs). Un opérateur est un symbole d'opération qui permet d'agir sur des variables ou de faire des "calculs".

5.1.1. Opérateurs numériques

Ce sont les quatre opérations arithmétiques tout ce qu'il y a de classique.

+ : addition
- : soustraction
* : multiplication
/ : Division

Avec en plus pour les entiers div et mod, qui permettent respectivement de calculer une division entière et le reste de cette division. Mentionnons également le ^ qui signifie « puissance ». 45 au carré s'écrira donc 45 ^ 2.

5.1.2. Opérateurs logiques (ou booléens)

Il s'agit du ET, du OU, du NON et du Ou Exclusif XOR.

5.2. Opérandes

Un opérande est une entité (variable, constante ou expression) utilisée par un opérateur dans une expression.

5.3. Expression

Une expression est formée d'opérandes et d'opérateurs. Un opérande peut être une constante ou une variable. L'évaluation d'une expression repose sur les règles de priorité entre operateurs selon l'ordre (du plus prioritaire au moins prioritaire) suivant :

1) Operateur unaires appliqués à un seul opérande : non, -, +
2) Operateurs multiplicatifs : *, /, div, mod, et.
3) Operateurs additifs : +, -, ou
4) Operateurs de relation : <,<=,>,>=,=,≠

Remarque :

- Pour des opérateurs de même priorité, l'expression est évaluée de gauche à droite.
- S'il y a des parenthèses, on commence par évaluer les plus internes
- Pour donner le type d'une expression (arithmétique, logique, chaine de caractère, …), il faut vérifier sa syntaxe(la façon d'écriture)

5.3 Les opérations utilisables sur les types :

Les opérations utilisables sur le type entier sont :

- arithmétiques : +,-,*,/ (division réelle), div(la division entière), mod (le reste de la division entière)
- de relation : <,<=,>,>=,≠

Les opérations utilisables sur le type réel sont :

- arithmétiques : +,-,*, /
- de relation : <,<=,>,>=,≠

Les opérations utilisables sur le type booléen(logique) sont :

- logique :et, ou, non
- de relation : <,>,=,≠ (nous avons faux<vrai)
- les opérateurs **succ** et **pred** qui retournent respectivement successeur et le prédécesseur d'un booléen. Exp : succ(faux)=vrai et pred(vrai)=faux
- l'operateur **ord**, exp : ord(faux)=0 et ord(vrai)=1.

Les opérations utilisables sur le type caractère sont :

- de relation : <,<=,>,>=,≠ (car les caratères sont ordonnées selon un code machine tel que ASCII(American Standard Code for Information Interchange)).
- les opérateurs **succ** et **pred** qui retournent respectivement successeur et le prédécesseur d'un booléen. Exp : succ('A')='B' et pred('C')='B'
- l'operateur **ord et chr :** ord('A')=65 et chr(65)='A' (si la machine utilise le code ASCII)

Les opérations utilisables sur le type chaine de caractères sont :

- de relation : <,<=,>,>=,≠ (car les caratères sont ordonnées)
- la fonction **concat** sert à concaténer 2 chaines. Exp : concat ('Tia','ret')
- la fonction **length** qui fournit la longueur d'une chaine.Exp : length('Tiaret')=6

Exemple :

```
Var
   i,j,k : entier ;
   x,y,z : réel ;
   A,B : logique ;
   c: caractère ;
   ch1, ch2 : chaine de caractère
```

Evaluer et donner le type des différentes expressions suivantes :

- -k-j div 2 expression correcte
- i mod x + y expression erronée : **mod** n'est pas valide sur le type réel
- i et non A<k+j expression erronée : **et** doit s'appliquer sur deux opérandes logiques
- c='e' ou c='f' expression erronée : **ou** n'est pas valide sur le type caractère
- (c='e') ou (c='f') expression correcte
- Length(concat(ch1,ch2)) expression correcte

6. Instructions de base

Une instruction est une action élémentaire commandant à la machine un calcul, ou une communication avec l'un de ses périphériques d'entrées ou de sorties. Les instructions de base sont : l'affectation et les instructions d'entrée sorties.

6.1. Affectation :

C'est l'action qui consiste à attribuer une valeur à une variable V (ayant un espace mémoire), l'affectation est notée par le signe ←. cette action est notée : V ← E(attribue la valeur de **E** à la variable **V :**

- ✓ **E** peut être une valeur, une autre variable ou une expression
- ✓ **V** et **E** doivent être de même type ou de types compatibles
- ✓ L'affectation ne modifie que ce qui est à gauche de la flèche

Syntaxe

Affectation d'une valeur à une variable : **Nom-variable ← valeur ;**
Affectation d'une variable à une variable : **Nom-variable1 ← Nom-variable2 ;**
Affectation du résultat de calcul à une variable: **Nom-variable ← nom-variable1 opérateur nom-variable2;**
Ou encore l'affectation du résultat de plusieurs calcul à une variable :
Nom-variable ← nom-variable1 opérateur1 nom-variable2 … opérateur-n nom-variable n ;

Exemple : A ← ((A+B)*2)

Variable	Avant l'action	Après l'action
A	2	10
B	3	3

6.2. Instructions d'entrée sorties

Au moment de l'exécution de l'algorithme, l'utilisateur et la machine ont besoin d'échanger des données ou des informations, on parle dans ce cas des instructions d'entrées (lecture) et de sorties (écriture). Lors du fonctionnement de l'algorithme et quand on arrive à une instruction de lecture ce dernier s'arrête à cette instruction et ne se poursuit que lorsque l'utilisateur a entré une valeur, et l'algorithme de sa part, affiche à l'utilisateur les résultats de son travail à travers les instructions d'écriture.

6.2.1. La lecture

L'instruction **lire (…)** permet à l'ordinateur d'acquérir des données à une variable déclarée (case mémoire bien définie) à partir de l'utilisateur au moyen d'un organe d'entrée (généralement le clavier).

Syntaxe : Lire (nom_variable) : la lecture d'une variable.

Exemple : Lire(a) On demande à l'utilisateur d'introduire une valeur pour a.
Lire(a,b,c) On demande à l'utilisateur d'introduire 3 valeurs pour a, b et c respectivement.

6.2.2. L'écriture

L'instruction **écrire (…)** permet de communiquer un résultat ou un message à l'utilisateur par unité de sortie

(écran).
Syntaxe : Ecrire (expression)
Exemples d'écriture
Ecrire('bonjour') :Affiche le message bonjour (constante)
Ecrire(12) : Affiche la valeur 12
Ecrire(a,b,c) : Affiche les valeurs de a, b et c
Ecrire(a+b) : Affiche la valeur de a+b
Ecrire('Somme=,' a+b) Affiche le message Somme= la résultat de l'expression a+b.

Construction d'un algorithme simple

Un algorithme se présente en général sous la forme suivante :
Déclaration des variables : On décrit dans le détail les éléments que l'on va utiliser dans l'algorithme (variables, constantes et structures),
Initialisation ou Entrée des données : On récupère les données et/ou on les initialise (par lecture et par affectation),
Traitement des données : On effectue les opérations nécessaires pour répondre au problème posé,
Sortie : On affiche les résultats (par l'écriture)
On rappelle que la structure d'un algorithme subit la forme suivante :

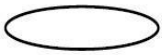
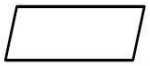
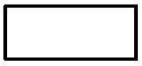
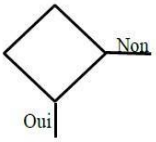
Algorithme Nom_de_l'algorithme	
Constante	
Variable	***Entête***
Liste des variables	
Début	
Action 1	
Action 2	**Corps**
…….	
Action n	
Fin	

Où la déclaration des variables se fait dans l'entête de l'algorithme et les autres tâches (initialisation, traitement et affichage) se font dans le corps de l'algorithme.
Exemple : calcul de la somme de deux entiers

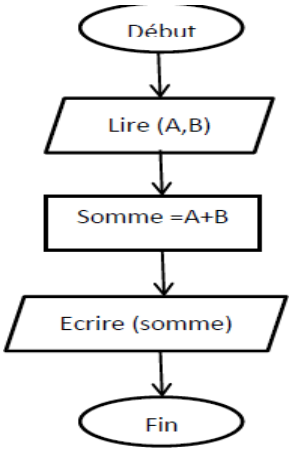
Algorithme addition ;
Var A, B, Somme : entier ;
Début
Lire (A) ;
Lire (B) ;
Somme ← A+B ;
Ecrire ('La somme=',Somme) ;
Fin.

7. Représentation d'un algorithme par un organigramme

Un **organigramme** (parfois appelé **algorigramme**) est une représentation graphique d'un algorithme. En général, on peut représenter un algorithme sous forme structurée ou sous forme graphique. Les opérations dans un organigramme sont représentées par les symboles dont les formes sont normalisées. Ces symboles sont reliés entre eux par des lignes fléchées qui indiquent le chemin.

	Début et fin d'un organigramme : on l'utilise pour commencer et pour terminer un algorithme.
	Entrée/ Sortie : on l'utilise pour lire ou écrire des données.
	Traitement : on l'utilise pour le reste des opérations hors la lecture et l'écriture.
	Choix avec condition : on l'utilise pour l'exploitation de conditions variables impliquant le choix d'une voie parmi plusieurs. Symbole couramment utilisé pour représenter une décision ou un aiguillage

Exemple :
L'algorithme de l'exemple précédent (somme de deux entiers) peut être représenté sous l'organigramme suivant :



8. Traduction d'un Algorithme en Langage C :

Introduction :
C est un langage de programmation impératif généraliste. Inventé au début des années 1970 pour réécrire UNIX, C est devenu un des langages les plus utilisés. De nombreux langages plus modernes comme C++, C#, Java et PHP ont une syntaxe similaire au C et reprennent en parti sa logique.
Il existe de nombreux langages de programmation de haut niveau comme le C, le Pascal, ou le Basic. Ils sont tous excellents et conviennent pour la plupart des tâches de programmation. Toutefois, les professionnels placent le langage C en tête de liste pour plusieurs raisons :

- Il est souple et puissant. Le langage C est utilisé pour des projets aussi variés que des systèmes d'exploitation, des traitements de textes, des graphiques, des tableurs ou même des compilateurs pour d'autres langages ;
- Disponible pour tous les types de processeurs et de systèmes d'exploitation ;
- Le langage C contient peu de mots. Une poignée d'expressions appelées mots clés servent de bases pour l'élaboration des fonctions ;
- il a influencé de nombreux langages plus récents dont C++, Java, C# et PHP ; sa syntaxe en particulier est largement reprise ;
- Le langage C est modulaire. Son code peut (et devrait) être écrit sous forme de sous-programmes appelés fonctions. Ces fonctions peuvent être réutilisées pour d'autres applications ou programmes.

9.1 Structure générale d'un programme C :

inclure des bibliothèques (#include)

définir des constantes (#define)
définir des types
définir des fonctions

int main()
{ <déclaration des variables locales du programme principal>
<instructions>
}

Un programme C est un ensemble de fonctions. Tout programme comporte au moins une fonction principale désignée par main.

A. La fonction main

La fonction **main** est la fonction principale des programmes en C. Elle se trouve obligatoirement dans tous les programmes. L'exécution d'un programme entraîne automatiquement l'appel de la fonction **main**.

B. Les commentaires

Un commentaire commence toujours par les deux symboles '**/***' et se termine par les symboles '***/**'. **Exemples :**
/* Ceci est un commentaire correct */

C. Utilisation des bibliothèques de fonctions

Pour pouvoir les utiliser, il faut inclure des fichiers en-tête (extension **.H**) dans les programmes par L'instruction **#include.**

Exemple :

```
#include<stdio.h>
Int main()
{ /* ce programme affiche le message « Bonjour » */
printf("Bonjour") ;
return 0 ;
}
```

9.2 Les types prédéfinis

A) Les entiers

définition	description	domaine min	domaine max	nombre d'octets
short	entier court	-32768	32767	2
Int	entier standard	-2147483648	2147483647	4
Long	entier long	-2147483648	2147483647	4
unsigned short	entier court	0	65535	2
unsigned int	entier standard	0	65535	2
unsigned long	entier long	0	4294967295	4

B) Les réels

définition	précision	mantisse	domaine min	domaine max	nombre d'octets
Float	simple	6	$3.4 * 10^{-38}$	$3.4 * 10^{38}$	4
Double	double	15	$1.7 * 10^{-308}$	$1.7 * 10^{308}$	8
long double	suppl.	19	$3.4 * 10^{-4932}$	$1.1 * 10^{4932}$	10

C) Les caractères

Définition	description	domaine min	domaine max	nombre d'octets
Char	Caractère	-128	127	1
unsigned char	Caractère	0	255	1

9.3 La définition des constantes :

On défini des constantes en utilisant la directive **define** ou le mot clé **const**.
Syntaxe : #define <idf> valeur ou bien const <idf> = valeur
Exemple : #define max 100 ou bien const max=100

9.4 La déclaration des variables :

<Type> <NomVar1>, <NomVar2>,..., <NomVarN>;
Exemples :
int compteur,X,Y; float hauteur,largeur; double M; char C;

Initialisation des variables

En C, il est possible d'initialiser les variables lors de leur déclaration:

int MAX = 1023; float X = 1.05;

9.5 Les opérateurs standards

A. Opérateurs arithmétiques

+	addition
-	soustraction
*	multiplication
/	division (entière et rationnelle !)
%	modulo (reste d'une div. entière)

B. Opérateurs logiques

&&	et logique (and)
||	ou logique (or)
!	négation logique (not)

C. Opérateurs de comparaison

==	égal à
!=	différent de
<, <=, >, >=	plus petit que, ...

D. Opérateurs d'affectation

=	Affectation ordinaire	X=Y
+=	ajouter à	X+=Y X=X+Y
-=	diminuer de	X-=Y X=X-Y
*=	multiplier par	X*=Y X=X*Y
/=	diviser par	X/=Y X=X/Y
%=	modulo	X%=Y X=X%Y
--	Décrémentation de 1	X-- X=X-1
++	Incrémentation de 1	X++ X=X+1
X = i++	passe d'abord la valeur de i à X et incrémente après	
X = i--	passe d'abord la valeur de i à X et décrémente après	
X = ++i	incrémente d'abord et passe la valeur incrémentée à X	
X = --i	décrémente d'abord et passe la valeur décrémentée à X	

9.6 Lecture

La lecture en langage C s'effectue avec l'utilisation de la fonction scanf qui permet de saisir des données au clavier et de les stocker aux adresses spécifiées par les arguments de la fonction.

Scanf ("chaîne de contrôle", argument-1,..., argument-n)

Exemple : Int x ; scanf ("%d",&x);

9 .7 Ecriture

L'écriture en C s'effectue à travers la fonction **printf**.

Sa syntaxe est : printf ("chaîne de contrôle ", expression-1, ..., expression-n);

Exemple: Int x,y,s ; s=x+y ; printf ("La some = %d \n",s);

Exemple:

```
#include <stdio.h> #include <math.h> main()
{
const float Pi=3.14159; float R,S,C ;
printf (" Entrez le rayon du cercle: "); scanf ("%f",&R);
S=Pi* (pow (R,2)); C=2*Pi*R;
printf ("La circonference du cercle = %f\n",C);
}
```

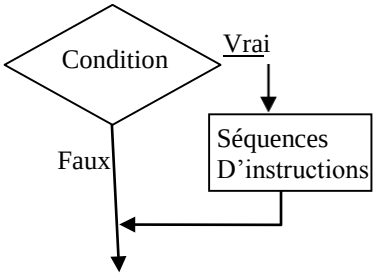
Chapitre 3 : Les structures conditionnelles (en langage algorithmique et en C)

Introduction

Les instructions Structurées (ou structures de contrôle) sont des instructions composées qui permettent de spécifier des traitements complexes, à partir d'instructions élémentaires, et d'exprimer la façon dont s'enchainent ces instructions. Il existe trois classes de ces instructions, à savoir : la séquence (l'enchainement naturel), l'alternative (l'enchainement conditionnel) et l'itération (l'enchainement répété).
Dans le cas des structures conditionnelles (alternatives) L'exécution d'un bloc d'instructions peut être conditionnée par la vérification d'une condition.
Il existe plusieurs formes de structure de contrôle :

1. Structure conditionnelle simple

Une structure de contrôle conditionnelle est dite à forme simple (réduite) lorsque le traitement dépend d'une condition. Si la condition est évaluée à « vrai », le traitement est exécuté. Dans cette structure la non- satisfaction de la condition ne correspond à aucune action à faire.

Organigramme	Algorithme	Langage C
	**Si** (condition) **Alors** Instruction 1 **Fin Si** **Si** condition **Alors** Instruction 1 Instruction 2 …. Instruction N **FinSi**	**If** (condition) Instruction 1 ; **If** (expression) **{** Instruction 1 ; Instruction 2 ; …. Instruction N ; **}**

Remarque: En langage C, si on ne met pas les accolades, le compilateur va considérer uniquement la première instruction comme instruction subordonnée à la structure if.
Exemple : Ecrire un algorithme qui affiche la valeur absolue d'un nombre entier A

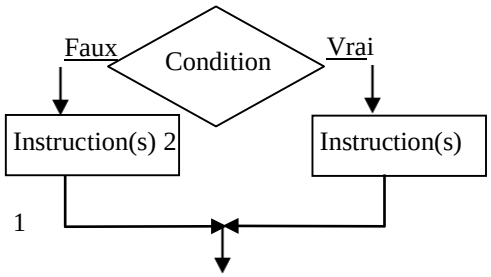
Solution en LA	Solution en Langage C

Algorithme ValAbs ;
Var A : entier ;
Début
Lire(A)
SI (A<0) **ALORS**
A ← A*(-1);
FSI
Ecrire ('la valeur absolue est :', A) ;
Fin
	#include<stdio.h>
	int main()
	{
	int x;
	printf("Donner A: ");
	scanf("%d",&A);
	if (A<0)
	A= -A;
	printf("la valeur absolue = %d \n",A);
	return 0 ;
	}

2. Structure conditionnelle composée

Une structure de contrôle conditionnelle est dite à forme alternative lorsque le traitement dépend d'une condition à deux états : Si la condition est évaluée à « vrai », le premier traitement est exécuté ; Si la condition est évaluée à « faux », le second traitement est exécuté.

Organigramme	Algorithme	Langage C

	**Si** (condition) **Alors**
	Instruction 1
	Sinon
	Instruction 2
	Fin Si
	
	Si (condition) **Alors**
	Instruction 1
	Instruction 2
	else
	Instruction 3
	Instruction 4
	Fin Si
		If (condition)
		Instruction 1 ; **else**
		Instruction 2 ;
		
		If (condition)
		{
		Instruction 1 ;
		Instruction 2 ;
		}
		else
		{
		Instruction 3 ;
		Instruction 4 ;
		}

Exemple : Ecrire un algorithme qui détermine si le nombre entier introduit est pair ou impair.

Solution en LA	Solution en Langage C

Algorithme pair_impair ;
Var a : entier ;
Début
Lire (a) ;
Si (a mod 2 = 0) **alors**
Ecrire (a,'est pair') ;
Sinon
Ecrire (a,' est impair') ;
Fin si
Fin
	#include<stdio.h>
	int main()
	{
	int a;
	printf("Donner a: ");
	scanf("%d",&a);
	if (a%2 == 0)
	printf("%d est pair \n", a);
	else
	printf("%d est impair \n",a);
	return 0 ;
	}

3. Structures conditionnelles imbriquées

Une structure de contrôle conditionnelle est dite imbriquée lorsqu'elle permet de résoudre des problèmes comportant plus de deux traitements en fonction des conditions. L'exécution d'un traitement entraîne automatiquement la non-exécution des autres traitements.

Algorithme	Langage C
Si (condition 1) **Alors** Traitement 1 **Sinon** **Si** (condition 2) **Alors** Traitement 2 **Sinon** **Si** (condition 3) **Alors** Traitement 3 **Sinon** ……. **Sinon** **Si** (condition n-1) **Alors** Traitement n-1 **Sinon** Traitement n **Fin Si** **Fin Si** **Fin Si** **Fin Si** **Fin Si**	**If** (condition) Traitement 1 ; **else if** (condition 2) Traitement 2 ; **else if** (condition 3) Traitement 3 ; **else if** …… **else if** (condition n-1) Traitement n-1 ; **else** Traitement n ;

Exemple : Ecrire un algorithme qui fait la comparaison de deux nombres entiers positifs N1 et N2.

Solution en LA	**Solution en Langage C**

Algorithme Comparaison ;
Var A,B: Entier;
Début
Lire (A) ;
Lire (B) ;
Si (A>B) **Alors**
Ecrire (‘A est plus grand que B’) ;
Sinon
Si (A<B) **Alors**
Ecrire (‘A est plus petit que B’) ;
Sinon
Ecrire (‘A égale à B’) ;
Fin si
Fin si
Fin
	#include <stdio.h>
	main()
	{
	int A,B;
	printf("Entrez deux nombres entiers :");
	scanf("%d %d", &A, &B);
	if (A > B)
	printf("%d est plus grand que %d\n", A,B);
	else if (A < B)
	printf("%d est plus petit que %d\n", A,B);
	else
	printf("%d est égal à %d\n", A, B);
	return 0;
	}

4. Structure conditionnelle de choix multiple

Une structure de contrôle conditionnelle est dite à choix lorsque le traitement dépend de la valeur que prendra un sélecteur. Ce sélecteur est de type entier, caractère ou booléen.

Cette structure conditionnelle est appelée aussi sélective car elle sélectionne entre plusieurs choix à la fois, et non entre deux choix alternatifs (le cas de la structure Si..Sinon).

Algorithme	**Langage C**

Selon sélecteur Faire
Valeur 1 : Action 1
Valeur 2 : Action 2-1
Action 2-2
Action 2-n
Valeur3 : Action 3
Valeur 4: Action 4
…..
Valeur N : Action N
Sinon
Action R
FinSelon
	switch (selecteur)
	{
	case Valeur 1 : Action 1 ; break ;
	case Valeur 2 : Action 2-1 ;
	Action 2-2 ;
	Action 2-n ; break ;
	case Valeur3 : Action 3;
	case Valeur 4 : Action4 ; break ;
	……..
	case Valeur N : Action N ; break ;
	default :
	Action R ;
	}

Exemple : Ecrire un algorithme qui affiche les mois de l’année.

Solution en LA	**Solution en Langage C**

Algorithme Mois_Annee ;
Var mois: Entier;
Début
Selon mois **Faire**
1 : Ecrire (‘Janvier’) ;
2 : Ecrire (‘Février’) ;
…..
12 : Ecrire (‘Décembre’)
Fselon
Fin
	#include<stdio.h>
	int main()
	{
	int mois;
	printf("Donner le Numéro du Mois: ");
	scanf("%d",&mois);
	switch (mois)
	{
	case 1: printf ("Janvier \n"); break;
	case 2: printf ("Février \n"); break;
	……..
	case 12:printf ("Décembre \n"); break;
	}
	}

5- Le branchement (sauts) :

Une instruction de branchement nous permet de sauter à un endroit du programme et continuer l'exécution à partir de cet endroit. Pour réaliser un branchement il faut tout d'abord indiquer la cible du branchement via une étiquette <etiq>. Ensuite on saute à cet endroit par l'instruction aller à <etiq> (go to <etiq> en C).

Algorithme	**Langage C**
Aller à <etiq> Action 1 Action 2 . Action N-2 <etiq> : Action N-1 Action N Action N+1	goto <etiq> ; Action 1 ; Action 2 ; . Action N-2 ; etiq : Action N-1 ; Action N ; Action N+1 ;

Exemple 1 :

Solution en LA	**Solution en Langage C**

Algorithme Ex1 ;
Var x: Entier;
Début
Ecrire ('Donner x') ;
Lire(x) ;
Aller à B1
Ecrire(x+5) ;
B1 :Ecrire(x-5) ;
Fin
	#include<stdio.h>
	int main()
	{
	int x;
	printf("Donner x: ");
	scanf("%d",&x);
	goto B1
	printf("%d",x+5) ;
	B1 : printf("%d",x-5) ;
	Return 0 ;
	}

5.1 Le branchement inconditionnel: c'est un branchement sans condition, il n'appartient pas à un bloc (**Si** (cond) **Alors**). Voir l'exemple 1.

5.1 Le branchement conditionnel: c'est un branchement avec condition, il appartient pas à un bloc (**Si** (cond) **Alors**).

Exemple 2 :

Solution en LA	**Solution en Langage C**

Algorithme Ex2 ;
Var x: Entier;
Début
Ecrire ('Donner x') ;
Lire(x) ;
Si (X>10) Alors Aller à B1 ;
Sinon Aller à B2 ;
Fsi
B1 : x←x+5 ;
Aller à B3 ;
B2 : x←x-5 ;
B3:Ecrire(x) ;
Fin
	##include <stdio.h> #include <math.h> main()
	{
	const float Pi=3.14159; float R,S,C ;
	printf (" Entrez le rayon du cercle: "); scanf ("%f",&R);
	S=Pi* (pow (R,2)); C=2*Pi*R;
	printf ("La circonference du cercle = %f\n",C);
	}