Chapitre 2: L'algorithme séquentiel simple

1. Préface

Lorsque l'ordinateur a été inventé dans les années quarante du siècle dernier, il fonctionnait à l'aide des tubes électroniques, et sa programmation (écriture de commandes) se faisait en entrant une série de nombres composés de zéro (0) et un (1). C'était difficile pour les programmeurs. Mais avec la création du transistor, la taille de l'ordinateur est devenue beaucoup plus petite et ses capacités ont augmenté. Des langages de programmation plus faciles à utiliser ont été inventés et qui sont très similaires au langage humain. Ces langages sont appelés langages de haut niveau. Tout programme écrit dans ces langages peut être traduit en langage machine (0.1) rapidement et automatiquement à l'aide du compilateur.

Dans ce chapitre, nous aborderons le concept de langage, la structure de l'algorithme, le concept de variables et de constantes, et nous verrons également quelques types d'instructions simples. Telles que: les instructions d'affectations, d'entrée et de sortie.

2. Notion de langage et langage algorithmique

2.1. Langage

C'est un moyen de communication et de compréhension entre les êtres humains. Dans le cas d'un ordinateur, c'est la façon dont un ordinateur comprend les commandes humaines. La langue se compose d'alphabet, de symboles, de vocabulaire, de règles grammaticales et de significations.

- L'alphabet: C'est un ensemble de lettres, chiffres et symboles.
- vocabulaire: ensemble de symboles et de mots, qu'il s'agisse de mots réservés ou de mots définis par le programmeur.
- la syntaxe: règles ou lois qui contrôlent le groupement et le placement des symboles et du vocabulaire.
- la sémantique: Elle précise le sens de chacune des instructions qui peuvent être construites dans le langage, et notamment ce qu'elle va produire à l'exécution.

2.2. Langage de Programmation

Un langage de programmation nous fournit un cadre pour développer des algorithmes et produire des programmes qu'un ordinateur peut exécuter. Elle permet notamment de décrire les structures de données qui seront traitées dans l'ordinateur, et les opérations qui seront réalisées. C'est un langage intermédiaire entre le langage humain et le langage machine. Là où un humain peut le comprendre, et un ordinateur peut le traduire dans un langage 0 et 1 qu'il comprend.

Il existe de nombreux langages de programmation, chacun avec ses propres avantages et règles, qui le rendent adapté à des degrés divers à un type particulier de logiciel. Le programmeur doit en connaître certaines et savoir quel langage est approprié pour chaque type d'application.

2.2.1. les types de langages de programmation

Il peut être divisé en deux parties

- Interprété: Si le programme n'est pas entièrement traduit en langage machine, mais plutôt traduit et exécuté instruction par instruction. Tels que: Matlab et les langages du Web.
- compilé: si le programme est entièrement traduit en langage machine avant son exécution. Tels que: C

Exemples

- C et C++: qui sera utilisé dans ce cours, et qui est considéré comme le langage mère de beaucoup d'autres langages.
- Langage Pascal: proche des algorithmes et utilisé dans les milieux pédagogiques.
- Delphi et WinDev: bons pour développer des logiciels de gestion.
- Langage Java: bon pour les applications réseau et mobiles
- C# (C-Sharp) et Visual Basic: utiles pour le développement de logiciels spécifiques à l'environnement Windows.
- Objectif C (Xcode): Développement de logiciels pour les produits Apple (Mac, iPad et iPhone).
- Langage PHP: spécifique au développement de sites web
- Matlab et Python: spécifiques à l'analyse de données, et destinés aux ingénieurs.

2.2.2. Compilateur

Programme informatique qui convertit le code source écrit dans un langage de programmation particulier en code ciblé qui peut être exécuté directement par un ordinateur.

Il existe de nombreux langages de programmation qu'un programmeur ne peut pas tous connaître. Pour que les programmeurs puissent travailler en équipe et échanger des solutions entre eux, ils doivent formuler ces solutions dans le langage algorithmique. Le langage algorithmique est le langage commun de tous les programmeurs.

2.2.3. Environnement de développement intégré

Pour écrire un programme, vous pouvez utiliser n'importe quel éditeur de texte, tel que le Bloc-notes. Mais cette méthode rend le processus de développement très difficile. Il existe donc un ensemble de programmes qui fournissent tous les outils nécessaires au processus de développement appelé IDE (Integrated development environment). L'IDE fournit tous les outils dont nous avons besoin pour concevoir, développer, tester, déboguer et déployer des applications, ce qui facilite et accélère le développement. L'IDE inclut tous les outils nécessaires pour commencer à concevoir des applications. comme:

- Éditeur de code: pour écrire et éditer le code du programme. Il effectue le formatage automatique, ce qui facilite le processus de lecture.
- Gestionnaire de projet: Pour gérer les fichiers qui composent un seul projet.
- Débogueur: détecte et corrige les erreurs dans le code.
- Raccourcis pour compiler et exécuter le programme.
- D'autres outils...

exemple: Dev-C++, Embarcadero, visual studio…

2.3. Eléments syntaxiques de base

2.3.1. Mots réservés

Ce sont des mots qui ont une signification préexistante pour le langage de programmation et qui ne peuvent pas être utilisés par le programmeur pour créer eux-mêmes de nouveaux éléments. Chaque langage a ses propres mots, comme "algorithm", "début" et "fin" dans l'algorithme, et "if" et "while" en C.

2.3.2. valeurs

Ce sont des nombres, des caractères (toujours entre guillemets simples ' '), des chaînes de caractères (toujours entre guillemets doubles " "), vrai et faux. Voir les types.

exemple: -2, 7, 3.12, 6e-7, 'k', 'خ', '1', '!', "azety", "سلام", vrai, faux

2.3.3. L'identificateur

C'est le nom que le programmeur donne à tout élément de l'algorithme qu'il veut créer. Tels que: nom de l'algorithme, nom de la variable, nom du type, nom de constante, nom de la fonction... Il a des règles et des conditions dans le langage C que nous adopterons dans l'algorithme.

Règles pour nommer les identificateurs:

- Le nom de l'identificateur ne peut contenir que des symboles littéraux et numériques de A à Z, de a à z et de 0 à 9, et le symbole « _ » trait de soulignement (tiret du huit)
- Il doit s'agir d'un seul mot, c'est-à-dire que le nom ne peut pas contenir d'espace " ".
- doit commencer par une lettre ou « _ », et non par un chiffre.
- ne doit pas être un mot réservé.
- L'identifiant doit être unique, il n'est pas possible de définir plus d'un élément avec le même nom.
- Il est recommandé d'utiliser des noms significatifs, par exemple nous utilisons Largeur au lieu de x.

Exemples

identificateurs **valides**

x, pi, Mat_info, estVide ,n5, _debut, _0a (Il vaut mieux l'évite)

identificateurs non valides

α, ج, π, é ,élève (Symboles inacceptables)

3a (Commence par un chiffre)

Mat info (La présence du l'espace)

debut, fin (mots réservés)

2.3.4. opérations

- **Opérations arithmétiques**

Opération	C	observation	exemple
- +	- +	signe	+3 -7 -a
- +	- +	Les deux opérandes entiers, Le résultat est un entier. l'un est réel, résultat est un nombre réel.	5+3.0 réel
*	*	Pour le produit, comme l'addition et la soustraction.	5*3 entier
/	/	pour la division réelle. Le résultat est toujours un nombre réel	5/3 ou 5.0/3 en C
mod	%	Pour calculer le reste de la division. Les deux opérandes sont entiers Le résultat est toujours un entier.	5 mod 3 ou 5%3 en C
div	/	Pour calculer le quotient (division entière). Comme le reste	5 div 3 ou 5/3 en C
^		Pour calculer la puissance, en C, on utilise la fonction pow(), le résultat est un nombre réel	5^2 ou pow(5,2) en C
√		Pour calculer la racine, en C, on utilise la fonction sqrt(), le résultat est un nombre réel	√5 ou sqrt(5) en C

- **Opérateurs relationnels**

Opération	C	observation
>, >=, <, <=		les même en C
=	==	En C « == » deux fois = est lu égal, tandis que « = » est utilisé pour l'affectation et lu reçoit.
≠	!=	lu défirent de

Le résultat de la comparaison est toujours de type logique booléen, c'est-à-dire vrai ou faux

- **Opérateurs booléens**

Opération		C	observation
non	négation	!	vrai si l'opérande est faux. et faux si vrai.
et		&&	vrai si les deux opérandes sont vrais, sinon faux
ou		||	fauxsi les deux opérandes sont faux, sinon vrai
xou	ou exclusif		vrai si l'un est vrai et l'autre est faux , sinon faux
=	équivalence	==	vrai si sont égaux.

- **Opérateurs de chaîne**

+ est utilisé pour concaténer deux chaines de caractère. exemple:

"bonne"+"Chance" donne "bonneChance" sans ajouter de l'espace. "bonne" + " " + "Chance" donne "bonne Chance"

- **la priorité**

Lors de l'évaluation d'une expression, nous suivons la priorité résumée dans le tableau suivant, et si la priorité est égale, la priorité est pour l'opération de gauche.

priorité	l'opération
0	()
1	+ et - de signe, non.
2	* / mod div
3	- +
4	>, >=, <, <=
5	≠, =
6	et
7	ou

Les parenthèses sont utilisées pour changer la priorité (et parfois pour la lisibilité).

2.3.5. Expression

C'est une structure de valeurs et d'identifiants, liée à l'aide des opérations, et lors de son évaluation, nous obtenons une seule valeur. Il est créé à l'aide de valeurs, de variables, de parenthèses et d'opérations.

Exemple: Supposons a=2, b=3 et ok=vrai

expression	résultat	expression	résultat	expression	résultat
5	5	a+3	5	ok	vrai
a	2	"In"+"fo"	Info	a*(b-7)>8 et ok	faux

2.3.6. Instruction

(statement) C'est une phrase ou une étape de la solution, c'est-à-dire la commande qui sera exécutée.

2.3.7. Bloc d'instructions

Il s'agit d'un ensemble d'instructions qui commence par le mot Début et se termine par Fin, ou commence par un mot réservé qui définit le début tel que si et se termine par le mot fin + le mot de départ tel que finSi.

En C, il commence par { et se termine par }.

exemple:

algorithme	C

Début
  Instruction 1
  …
  Instruction n
Fin
	{
	  Instruction 1 ;
	  …
	  Instruction n ;
	}

2.3.8. Les commentaires

Ce sont des textes qui sont ignorés lors de la traduction du programme et qui ne font pas partie de la solution (algorithme). Ils sont ajoutés aux programmes pour laisser des explications et faciliter la compréhension.

Les commentaires en C sont ajoutés à l'aide de « // » pour ajouter un commentaire d'une ligne. Il commence par // et se termine par un retour à la ligne.

Des commentaires peuvent également être ajoutés en commençant par « /* » et se terminant par « */ », qui peuvent s'étendre sur plusieurs lignes.

exemple:

```
// Commentaire d'une seule ligne
/* Commentaire
multiligne*/
```

2.4. Expression de l'algorithme

Un algorithme peut être exprimé par l'écriture dans le langage naturel. Tels que: arabe ou Français. Cependant, le langage naturel est ambigu et inexact. Nous écrivons donc l'algorithme en utilisant des Pseudocode, des organigrammes et des langages de programmation.

1- Pseudocode: Décrire l'algorithme dans les langages humains tels que l'arabe, le Français ou l'anglais de la même manière que les langages de programmation. Certains utilisent beaucoup de détails (pour se rapprocher des langages de programmation), d'autres en utilisent peu (c'est-à-dire plus proches du langage humain)... Il n'y a pas de règle spécifique pour écrire ce type de code.

2- Structure organisationnelle: Il s'agit d'une représentation illustrée de l'algorithme qui montre les étapes pour résoudre le problème du début à la fin, tout en masquant les détails pour donner une image générale de la solution. Les flèches et les formes géométriques convenues sont utilisées pour représenter les étapes.

3- Code: où l'algorithme est écrit dans un langage de programmation directement tel que C, auquel cas l'ordinateur peut le traduire en langage binaire, pour être exécuté directement par le processeur.

3. Parties d'un algorithme

Un algorithme est constitué d'un ensemble de données et d'un ensemble d'instructions sur ces données. L'algorithme est très similaire dans sa structure à une recette de cuisine. Généralement, la recette se compose: du titre de la recette, puis des ingrédients, et enfin, de la méthode de préparation.

il prend la forme suivante:

```
Algorithme nom
  Déclaration des données dont nous avons besoin
Début
  instructions
Fin
```

L'algorithme se compose de trois parties de base:

- en-tête: Il est composé du mot Algorithme, suivi du nom qui explique le problème à résoudre. Il doit s'agir d'un identifiant valide.
- Déclarations: Dans lesquelles des places sont réservées en mémoire pour des données (constantes et variables), qui seront utilisées comme données et résultats.
- Instructions: un ensemble d'étapes ou de commandes qui seront exécutées lors de l'exécution de l'algorithme. Il commence par Début et se termine par Fin. Il y a 5 types principaux:
  1. Instruction d'affectation.
  2. Instruction de lecture (pour entrer des données).
  3. Instruction d'écriture (pour montrer les résultats).
  4. Instruction conditionnelle.
  5. Instruction itérative.

4. Les Données: variables et constantes

4.1. Constante

Il s'agit d'une valeur (chiffres ou symboles) qui a un nom et ne peut pas être modifiée pendant l'exécution du programme.

Déclaration des constantes:

```
Const Identificateur=valeur
```

Const ou **Constante**: Ce sont deux mots réservés, qui permettent la déclaration de constantes.

identificateur: est le nom qu'on donne a la constante.

Valeur: est la valeur qui est donnée à la constante.

exemple:

```
Const
     Pi= 3.1415926
     DEP  = "قسم الاعلام الالي"
```

Avantages des constantes

- Code rétracté où une grande phrase peut être remplacée par un petit mot. Tels que: P au lieu de 3.1415926.
- Évitez les erreurs en fournissant un nom significatif. Par exemple: PI au lieu de 3,1415926.
- Simplifiez la maintenance du code, de sorte que la valeur ne soit modifiée qu'à un seul endroit.

4.2. Variable

Emplacement dans la mémoire utilisé pour stocker des données. Il a un nom, un type et une valeur (a une adresse dans le deuxième semestre).

- **Nom**: identifiant utilisé par le programmeur pour revenir à la valeur stockée et pour manipuler la variable. **Ex**: poids.
- **Type**: Tout dans l'ordinateur est 0 et 1. Le type détermine la façon dont il est traduit, ainsi que la taille à réserver en mémoire. C'est-à-dire le nombre de bits et les opérations autorisées. Exemple: int (32 bits).
- **Valeur**: C'est le contenu des bits qui le composent, c'est-à-dire sa valeur. Habituellement, c'est la chose qui change pendant l'exécution du programme. Tel que:
  1101 qui représente le chiffre 13, ou le chiffre -5 si l'on considère le 1 à gauche comme le signe "-".

Déclaration des variables:

```
Var Identificateur: Type
```

Var ou **Variable**: Ce sont deux mots réservés qui permettent la déclaration de variables.

Identificateur: Le nom qu'on donne à la variable.

Type: Le type de la variable.

"," peut être utilisé pour déclarer plusieurs variables du même type.

Exemples:

```
Var
    age: entier
    sexe: caractères
    x, y, z: réel
```

Remarque: Par convention, les noms des constantes sont en MAJUSCULES et les noms des variables en minuscules.

5. Types de donnés

Type: est le domaine auquel appartiennent les données. tel que les nombres, textes, images, audio ou vidéo. Le type détermine comment les bits, qui composent la variable, sont traduits et la taille à réserver sur la mémoire, c'est-à-dire le nombre de bits et les opérations autorisées. Lorsque vous définissez une variable, vous devez spécifier son type. Il existe 5 types de base dans l'algorithme:

1. Entier Integer comme:-5, 0, 1, 13
2. Réel Real: -7, 0, 1, 3.14, 2.7e03
3. Booléen Boolean ne contient que vrai ou faux.
4. Caractère Charactere: Il comprend toutes les icônes sur le clavier. Tels que: chiffres, lettres dans toutes les langues et symboles imprimés (visuels) et non imprimés. Ils sont toujours entre guillemets simples (virgule supérieure). Tels que: 'a', 'M', '1', '+', ',', 'س'.
5. Chaines de caractères, String: Un ensemble de symboles, de longueur 0 ou plus, toujours entre guillemets doubles. Tels que: "informatique", "Bonne chance\n", "1", "3.14".

Remarques:

- Nous utilisons « . » au lieu de « , » pour exprimer les nombres décimaux.
- 1 n'est pas la même chose que 1., pas la même chose que '1', et pas la même chose que "1". Le premier est un entier, le second est un nombre réel, le troisième est un caractère et le dernier est une chaîne de caractère.
- 'a' n'est pas la même chose que "a". Le premier est un caractère et le second une chaîne de longueur 1.
- Les minuscules ne sont pas les mêmes que les majuscules. c'est-à-dire que 'a' n'est pas la même chose que 'A'.
- Certains symboles (touches) ne s'impriment pas. Par exemple: l'espace ' ', retour à la ligne '\n'.

- La barre oblique inverse, (\) antislash, est utilisée pour représenter visuellement certains symboles invisibles ou spéciaux. Par exemple: nouvelle ligne '\n' et tabulation c'est-à-dire un grand éspace '\t'. pour imprimer " nous utilisons \" et pour imprimer \ nous utilisons \\.
- Il y a une chaîne de caractère vide. C'est-à-dire qu'elle ne contient aucun caractère, sa longueur est de 0 et nous le symbolisons par "".

6. Instructions de base

6.1. Affectation

C'est le processus qui nous permet de stocker une valeur dans une variable.

Syntaxe:

```
variable←exp
```

variable: C'est le nom d'une variable.

exp: est une expression, (identificateurs, valeurs et opérations. voir 2.3.4), calculée pour obtenir une valeur unique, placée dans la variable.

← lu en Français reçoit, et en anglais gets. La flèche pointe toujours vers la variable.

Une variable ne peut porter qu'une seule valeur à chaque point de l'exécution du programme. Lorsque l'opération est effectuée, seule la variable de gauche change. Il perd l'ancienne valeur et prend la nouvelle. Pour que le processus d'affectation s'effectue correctement, la valeur de l'expression de droite et la variable de gauche doivent être du même type, ou du moins de types compatibles.

Exemple

`a←5`	a reçoit 5
`b←a*2`	b reçoit 10
`a←0`	a reçoit 0
`b←b-1`	b reçoit 9
`c←'b'`	c reçoit la lettre b
`d←b>a`	d reçoit vrai
`s←"name"`	s reçoit le mot nom

Avant qu'une variable puisse être utilisée, elle doit être déclarée et affectée d'une valeur initiale. Pour obtenir la valeur de n'importe quelle variable ou constante, il suffit d'écrire son nom.

6.2. Instructions d'Entrées/Sorties

Pour interagir avec l'utilisateur, le programmeur dispose de deux instructions: Lire() et Ecrire().

6.2.1. Entrée: Lire()

Lire() est une fonction prête à l'emploi dans les algorithmes. Où vous entrez une valeur de l'utilisateur, via le clavier, et l'affectez à la variable entre parenthèses. Il est toujours utilisé pour saisir des données.

Syntaxe:

```
Lire(variable)
```

variable: est le nom d'une variable. lire() ne peut être utilisé qu'avec des variables.

Lorsque le programme est exécuté et que l'instruction d'entrée lire() est rencontrée, l'exécution s'interrompt jusqu'à ce que l'utilisateur entre les données. Le processus de saisie se termine en appuyant sur la touche Entrée. Le programme continuera d'être exécuté. Lorsque le programme est exécuté et que l'instruction d'entrée lire() est rencontrée, l'exécution s'interrompt jusqu'à ce que l'utilisateur entre les données. Le processus de saisie se termine en appuyant sur la touche Entrée. Le programme poursuit son exécution.

Plusieurs variables peuvent être saisies à la fois, séparées par une virgule « , ». Dans ce cas, l'utilisateur saisit la valeur de la première variable, puis appuie sur la touche espace, puis saisit la valeur de la seconde variable, et n'appuie sur la touche entrée, qu'après avoir saisi la valeur de la dernière variable.

Exemple:

`Lire(nom)`	l'utilisateur entre un série de lettres, par exemple <Muhammad> puis appuie sur la touche entrée.
`Lire(a, b)`	Il saisit un nombre, par exemple « 15 », puis appuie sur espace, puis saisit le deuxième nombre, par exemple « 20 », puis appuie sur la touche entrée.

6.2.2. Sortie: Ecrire()

Ecrire() est une fonction prête à l'emploi dans l'algorithme. Elle affiche sur l'écran tout ce que nous mettons à l'intérieur de ses parenthèses. Elle est toujours utilisée pour imprimer les résultats.

Syntaxe:

```
Ecrire(exp)
```

ou

```
Ecrire("message")
```

exp: C'est une expression, calculée pour obtenir une seule valeur, à afficher à l'écran.

"message": est tout texte que vous souhaitez afficher tel quel à l'écran. Il n'est pas calculé. C'est dans n'importe quelle langue, ou ensemble de lettres que ce soit. Il doit être entre guillemets doubles, qui ne sont pas affichés à l'écran.

Plusieurs valeurs et textes peuvent être affichés à la fois, séparés par une virgule « , ».

Exemple:

`Ecrire(nom)`	La valeur de la variable nom apparaît à l'écran par exemple <Mohammed>
`a←5` `Ecrire(a+3)`	affiche 8 sans modifier la valeur de a.
`Ecrire("carré de ", a, " est ", a*a)`	affiche: carré de 5 est 25
`Ecrire("b=",a)`	affiche: b=5

Remarques

- toujours avant l'instruction Lire(), vient l'instruction Ecrire(), afin d'expliquer à l'utilisateur ce qui lui est demandé d'entrer.
- L'utilisateur écrit sur le clavier, tandis que le programme (ordinateur) lit <Lire()> à partir du clavier, et le programme écrit <Ecrire()> sur l'écran, tandis que l'utilisateur lit à partir de l'écran.
- on pout généraliser <Lire()> pour l'entrée de toutes les unités d'entrée, et <Ecrire()> pour la sortie de toutes les unités de sortie.

7. Construction d'un algorithme simple

Après avoir vu que l'algorithme se compose de 3 parties, à savoir: en-tête, la déclaration et les instructions. et nous avons appris à déclarer des constantes et des variables, et nous avons appris 3 types d'instructions, à savoir: affectation, lecture et écriture. Maintenant, nous pouvons écrire des algorithmes simples.

Pour connaître les variables, nous posons la question: « Quelles données sont nécessaires et quel est le résultat attendu ? » La partie instruction, généralement, comprend trois étapes de base:

- La première étape, « Inputs »: les données nécessaires à la mise en œuvre sont saisies. en utilisant l'instruction <Lire()>.
- La deuxième étape, « le traitement »: Elle contient un ensemble d'instructions nécessaires pour résoudre le problème. à l'aide de l'instruction d'affectation.
- La troisième étape, "Outputs": les résultats sont présentés. à l'aide de l'instruction <Ecrire()>.

Exemple 1:

Écrivez un algorithme qui calcule l'aire d'un cercle.

```
Algorithme surf_cercle
Const
     P=3.14
Var
   r, s:entier //r est le rayon et s est la surface
Début
   Ecrire("Entrer le rayon")
   Lire(r)
   s←p*r*r
   Ecrire("L'aire du cercle est:" , s)
Fin
```

Exemple 2:

Écrivez un algorithme qui calcule la moyenne pour ASD1.

```
Algorithme moy_ASD1
Var
   cont, td, tp, moy:réel
Début
   Ecrire("Saisir la note d'examen, de travaux dirigés et de travaux pratiques")
   Lire(cont, td, tp)
   moy←(cont*3+td+tp)/5
   Ecrire("La moyenne est:" , moy)
Fin
```

7.1. Exécution d'un algorithme

L'exécution de l'algorithme vise à connaître la valeur de chaque variable après chaque instruction. Où l'exécution commence du mot début au mot fin. Au début, les valeurs des variables sont indéfinies (non vides), puis après chaque affectation ou lecture, la valeur de la variable change.

Exemple

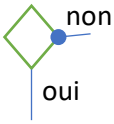
Algorithme miroir	Exécution		
Var			
a, b, c:entier	a	b	c
Début			
a←357	357	?	?
c←0	357	?	0
b←a mod 10	357	7	0
c←c*10+b	357	7	7

a←a div 10	35	7	7
b←a mod 10	35	5	7
c←c*10+b	35	5	75
a←a div 10	3	5	75
b←a mod 10	3	3	75
c←c*10+b	3	3	753
Fin			

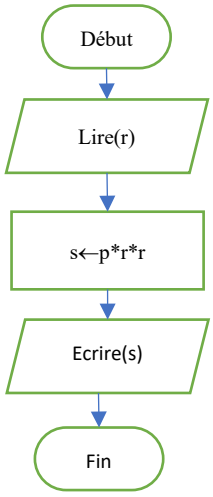
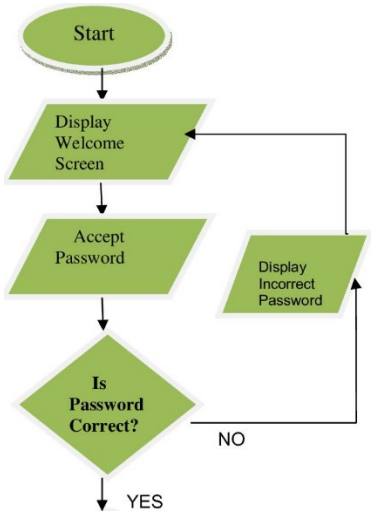
8. Représentation d'un algorithme par L'algorigramme

L'algorigramme est une représentation visuelle d'un algorithme avant sa programmation. Il nous montre la séquence des opérations, et nous donne la structure générale des composants de l'algorithme. L'algorigramme présente de nombreux avantages: Il permet de mieux visualiser les idées. Comme il est compris de tous, il permet de travailler plus facilement en équipe.

Un certain nombre de formes sont utilisées dans l'algorigramme, dont les plus importantes sont:

Symbole	utilisation
	Début, fin, interruption
	Entrée – Sortie lire()/écrire()
	Symboles De Traitement comme l'affectation
	Symboles Logiques: Choix avec condition

Exemple:

l'algorithme qui calcule l'aire d'un cercle	l'algorithme pour vérifier l'exactitude du mot de passe.
	

9. Traduction en langage C

"**C**" est un langage impératif de haut niveau entièrement compilé. C'est l'un des langages de programmation les plus utilisés au monde. Il est considéré comme la langue mère de nombreux langages de programmation. Dans ce cours, nous utiliserons Dev-C++ comme IDE. Nous notons ici que "C" est sensible à la casse, car il fait la distinction entre MAJUSCULES et minuscules . <main> n'est pas la même chose que <Main>, <MAIN> ou <mAin>. Par conséquent, nous vous recommandons de toujours écrire en minuscules.

Toutes les instructions simples (déclaration, affectation, E/S, retour) doivent se terminer par un point-virgule ";".

9.1. Le préprocesseur

Avant que le programme ne soit réellement compilé, les fichiers de code source sont traités par un préprocesseur, qui résout certaines des directives qui lui sont données. Ex: inclure d'autres fichiers (bibliothèques), remplacer des mots par d'autres phrases (macro).

Une directive donnée à un préprocesseur commence toujours par #.

9.1.1. #include

La directive #include indique au gestionnaire d'inclure le contenu d'un autre fichier dans le code du programme en cours.

```
#include <nom_de_fichier>
```

Généralement, ces fichiers sont des bibliothèques de fonctions définies et prêtes à l'emploi.

Exemple:

- Pour utiliser les fonctions I/O (scanf et printf), nous utilisons la librairie stdio.h.
- Pour utiliser les fonctions mathématiques (sin, cos, exp, pow, sqrt, ...), on utilise la bibliothèque math.h.
- Pour utiliser les fonctions de chaîne (strlen, ...), nous utilisons la bibliothèque string.h.

```
#include <stdio.h>
#include <math.h>
```

9.1.2. Macro

Une macro, dans sa forme la plus simple, est définie comme suit:
`#define nom_macro <le texte de remplacement>`

Exemple:
```
#define N 10
```

Le préprocesseur remplace toutes les occurrences du mot N par 10.

9.2. Les types

9.2.1. Types prédéfinis

Les tableaux suivants résument les types de base dans l'algorithme et leurs équivalents en C.

- Entiers dans l'algorithme de -∞ à +∞

Types	Taille en octets	Taille en bits	intervalle
`char`	1	8	-2^7 , 2^7-1
`short`	2	16	-2^{15} , $2^{15}-1$
`long`	4	32	-2^{31} , $2^{31}-1$
`int`	4	32	-2^{31} , $2^{31}-1$

- Les nombres naturels dans les algorithmes de 0 à +∞. Étant donné que les entiers contiennent des nombres naturels, nous utilisons généralement des entiers pour les exprimer. On peut également ajouter <unsigned> devant un type en C pour exprimer uniquement les nombres naturels.

Types	Taille en octets	Taille en bits	intervalle
`unsigned char`	1	8	0 , 2^8-1
`unsigned short`	2	16	0 , $2^{16}-1$
`unsigned long`	4	32	0 , $2^{32}-1$
`unsigned int`	4	32	0 , $2^{32}-1$

- Nombres réels

Types	Taille en octets	la précision	intervalle
`float`	4	6	
`double`	8	8	
`long double`	10	8	

- Type booléen: il n'y a pas de type booléen en C, mais int est utilisé à la place. où vrai est le nombre 1 et faux est 0. Tout nombre autre que 0 est traduit par vrai.
- Le type de caractères est char.
- Type chaîne de caractères: Pour exprimer la chaîne de caractères en C, on utilise des tables (Chapitre 5) char[] ou des pointeurs (second semestre) char*.

9.2.2. Remarques

- Le type int est le type générique des entiers.
- Le type char est utilisé pour les entiers et les caractères. Où chaque caractère est associé à un nombre.
- En C++ il y a le type **bool** pour le type booléen et **string** pour la chaîne de caractères.
- Dans ce cours, nous utilisons **char** pour les caractères, **int** pour les entiers et booléen, et **float** pour les nombres réels.

9.2.3. La conversion de type

- **Implicite**: C'est fait automatiquement par le compilateur. Ce passe du petit type au grand type sans perdre d'informations. Exemple: char en int, int en float ou float en double. Convertir 5 en float devient 5.0.

- **Explicite** (cast): Lorsque la conversion entraîne la possibilité de perdre des informations, il faut déclarer que c'est le programmeur qui fait l'opération, et il lui suffit de préciser le type de destination entre parenthèses devant l'expression pour le convertir. Par exemple: (int) 3.1416 pour qu'elle devienne 3.

9.2.4. Définition de nouveaux types

Pour créer un nouveau type ou changer le nom d'un type spécifique, nous utilisons le mot réservé **typedef**. Il prend la forme suivante:

```
typedef ancien_nom nouveau_nom ;
```

Exemple: Pour déclarer un nouveau type appelé Banane, dont l'origine est int, on utilise:

```
typedef int Banane;
```

9.3. Déclaration des variables et des constantes

9.3.1. Déclaration des variables

Syntaxe:

```
Type variable;
```

exemple:

```
int age;
char sexe;
float x, y, z;
Banane b;
```

9.3.2. Déclaration des constantes

Syntaxe:

```
const Type Identificateur=valeur;
ou
Type const Identificateur=valeur;
```

exemple:

```
int const N = 10;
const float Pl = 3.1415926;
const char[] DEP  = "قسم الاعلام الالي";
```

Remarque: Des macros peuvent être utilisées pour déclarer des constantes. Tel que:

```
#define DEP  "قسم الاعلام الالي"
```

9.4. L'affectation

Nous utilisons = à la place de ← et lisons reçoit et non égale.

Syntaxe:

```
variable = expression;
```

exemple:

a=5;	a reçoit 5
b=a*2;	b reçoit 10
a=0;	a reçoit 0
b=b-1;	b reçoit 9
c='b';	c reçoit la lettre b
d=b>a;	d reçoit 1
s="name";	s reçoit le mot nom

Déclaration avec initialisation

```
float x, y=3, z;//y est initialisé avec 3
```

Dans le cas où l'affectation apparaît plusieurs fois, la priorité est à droite. Tel que:

```
b=3 ;
a=b=5+3 ;
```

b prend 8, puis **a** prend la valeur de **b**, soit 8.

Raccourcis d'affectation en C.

<table>
<tr><th colspan="2">expression</th><th>observation</th><th>exemple</th></tr>
<tr><td colspan="2">v+=exp ; ⇔ v=v+(exp) ;</td><td rowspan="5">Les parenthèses sont importantes</td><td rowspan="5">x=2 ;
x*=5+3 ; ⇔ x=x*(5+3) ;
≠ x=x*5+3 ;
x devient 16</td></tr>
<tr><td colspan="2">v-=exp ; ⇔ v=v-(exp) ;</td></tr>
<tr><td colspan="2">v*=exp ; ⇔ v=v*(exp) ;</td></tr>
<tr><td colspan="2">v/=exp ; ⇔ v=v/(exp) ;</td></tr>
<tr><td colspan="2">v%=exp ; ⇔ v=v%(exp) ;</td></tr>
<tr><td>v++ ; ⇒ v=v+1 ;</td><td rowspan="2">Dans le cas où il est contenu dans une autre instruction, le calcul est effectué avec la valeur actuelle de la variable, et après achèvement, 1 est ajouté ou soustrait à la variable selon l'opération.</td><td rowspan="2">x=2 ;
y=3+x++ ;⇔ y=3+x ;x=x+1 ;
Autrement dit, y devient 5 et x 3</td></tr>
<tr><td>v-- ; ⇒ v=v-1 ;</td></tr>
<tr><td>++v; ⇒ v=v+1 ;</td><td rowspan="2">S'il est inclus dans une autre instruction, 1 est ajouté ou soustrait selon l'opération avant le calcul de l'expression, qui se fait avec la nouvelle valeur de la variable.</td><td rowspan="2">x=2 ;
y=3+ ++x ;⇔ x=x+1 ;
y=3+x ;
Autrement dit, y devient 6 et x 3</td></tr>
<tr><td>--v ; ⇒ v=v-1 ;</td></tr>
</table>

Observation

- v++ n'est pas identique à v+1, mais v=v+1.
- v++, ++v, v=v+1, v+=1 sont tous équivalents s'ils sont présentés dans une instruction séparée.
- La différence entre v++, ++v lorsqu'il apparaît dans une autre phrase, est l'ajout pré- ou postérieur.

Instruction vide: C'est une instruction qui ne fait rien, comme l'instruction point-virgule « ; » . et des instructions telles que i + 1, où le résultat est calculé et ignoré, sans aucun changement dans l'état du programme.

9.5. Entrée et sortie

9.5.1. printf (print formatted)

La fonction printf, définie dans la bibliothèque stdio.h, est utilisée pour écrire des données formatées dans l'unité de sortie standard, par défaut l'écran.

Syntaxe:

```
printf (format, expression_1,… , expression_n);
```

- format: est un texte ou une chaîne de caractères, qui s'affiche tel quel à l'écran. Sauf pour le symbole "%" pour exprimer les formats des expressions et le caractère "\" pour le symbole d'échappement.
- expression: calculée pour obtenir une seule valeur. qui va être afficher à l'écran dans le format <format>.

Le format prend la forme suivante:

```
% [flags] [width] [.prec] type_char
```

où il commence toujours par %

- flags

- -: Faire l'alignement à gauche.
- +: Afficher le signe du nombre.
- Vide: Afficher un espace au lieu de + dans le cas d'un nombre positif.

- width représente le plus petit nombre de chiffres à l'écran pour afficher une valeur spécifique, où si ce nombre est supérieur à la taille requise, la différence est remplie d'espaces ou de zéros, selon ce nombre s'il commence par 0 ou non.
- .prec Le nombre de chiffres après la virgule dans float.
- type_char Un caractère représentant le type de valeur à sortir. Le tableau suivant présente les formats les plus couramment utilisés.

format	utilisation
%d	Pour entrer ou sortir un nombre dans un système décimal (10)
%o	Pour entrer ou sortir un nombre dans le système octal (8)
%x	Pour entrer ou sortir un nombre dans le système hexadécimal (16)
%u	Pour entrer ou sortir un numéro naturel non signé **u**nsigned
%i	Pour entrer ou sortir un entier (int) comme %d
%f	Pour entrer ou sortir un réel (float)
%c	Pour entrer ou sortir un caractère (char)
%s	Pour entrer ou sortir une chaine (string, char*)
%e	Pour entrer ou sortir un nbr en format scientifique comme 3e-2

– Certains caractères sont spéciaux, nous devons donc utiliser une technique d'échappement pour les utiliser. En C, le caractère d'échappement est le (\), barre oblique inverse (antislash), et nous l'utilisons pour ajouter une nouvelle ligne '\n' ou une tabulation (un grand espace) '\t'. et pour imprimer " nous utilisons \", pour imprimer \ nous utilisons \\ et pour imprimer % nous utilisons %%.

exemple:

`printf("Bonjour");`	affiche Bonjour
`int a=13;` `printf("a=(%d)10\ta=(%o)8\ta=(%X)16\n",a ,a ,a);`	a=(13)10 a=(15)8 a=(D)16
`a=66;` `printf("a=%i\ta=%f\ta=%c\ta=%c\n",a ,a ,a ,a+32);`	a=66 a=0.000000 a=B a=b Parce que 66 n'est pas forcément 66 en float Et 66 si nous le voyons comme un caractère, il représente la lettre B tandis que 66 + 32 = 98 représente le codage de la lettre b en minuscules.
`float pi=3.1415926;` `printf("%f\t%.4f\t%06.2f" ,pi ,pi ,pi);`	3.141593 3.1416 003.14

9.5.2. scanf (scan formatted)

La fonction scanf, définie dans la bibliothèque stdio.h, est utilisée pour lire les données formatées sur l'unité d'entrée standard, par défaut le clavier. La fonction copie la valeur saisie par l'utilisateur à l'adresse de la variable (place en mémoire). Nous utilisons donc & avant le nom de la variable.

Syntaxe:

```
scanf(format, &variable_1, … , &variable_n);
```

format: Chaîne de caractères représentant le format de lecture.

variable: Représente le nom de la variable. Il est précédé par <&>, sauf s'il s'agit d'une variable de type chaîne (de type pointeur).

Le **format** prend la forme suivante:

% [width] **type_char**

- width:: Un nombre qui contrôle le nombre maximum de caractères à lire dans le champ de saisie actuel.
- type_char:: Un caractère représentant le type de valeur à entrer. Les mêmes symboles dans le tableau pour printf.

Exemple:

`scanf("%s", nom);`	& n'est pas utilisé pour lire une chaîne.
`scanf("%d%f",&a, &b);`	Saisit un nombre, puis appuie sur espace, puis saisit le second nombre, puis appuie sur la touche entrée.

Remarque: Il est recommandé d'entrer une seule valeur par instruction scanf.

problème de scanf avec les caractères et les chaînes.

Lors de la lecture d'un caractère à l'aide de scanf("%c"...), l'utilisateur saisit le premier caractère, puis appuie sur la touche Entrée, qui, à son tour produit le caractère '\n', qui n'est pas supprimé de la mémoire, et lorsque le programme rencontre l'instruction scanf("%c", ...) pour la deuxième fois, il n'attend pas ce que l'utilisateur entrera, mais assigne plutôt le caractère \n à la deuxième variable.

Pour éviter ce problème, dans le second scanf nous utilisons un espace ' ' après % comme ceci: scanf("% c"...). Ou nous utilisons la fonction **getch** définie dans la bibliothèque string.h.

Exemple: Nous allons essayer d'entrer 3 lettres a, b, c dans les variables c1, c2 et c3. Nous utilisons:

```
scanf("%c", &c1) ;
scanf("%c", &c2) ;
scanf("%c", &c3) ;
```

L'utilisateur appuie sur a, puis sur entrée. Le programme affecte a à c1 et \n à c2, et attend que l'utilisateur entre le deuxième caractère à affecter à c3. Pour éviter ce problème, nous utilisons:

```
scanf("%c", &c1) ;
scanf("% c", &c2) ;
scanf("% c", &c3) ;
```

Le problème de scanf avec des chaînes apparaît lorsque nous essayons d'insérer une chaîne contenant des espaces dans une variable. par exemple:

```
scanf("%s%s", v1,v2) ;
```

Lorsque nous entrons les mots math info et que nous appuyons sur Entrée, le programme attribue le premier mot à v1 et le deuxième mot à v2. Cependant, lorsque nous voulons insérer les deux mots (un nom composé, par exemple) dans une variable, nous utilisons:

```
scanf("%s", v1) ;
```

Lorsque nous entrons les mots math info et que nous appuyons sur Entrée, le programme attribue le premier mot à v1 et le deuxième mot est perdu.

Pour éviter ce problème, nous utilisons la fonction gets définie dans la bibliothèque string.h.

```
#include <sting.h>
gets(v1);
```

9.6. Structure d'un programme en C

```
1.  #include <stdio.h>
2.    Déclaration des constantes, des types et des variables
3.  int main()
4.  {
5.    Déclaration des constantes et des variables
6.    Les instructions
7.    return 0;
8.  }
```

l'explication:

1. Pour inclure la bibliothèque stdio.h qui contient scanf et printf.
2. Lieu des déclarations publiques.
3. main(): Chaque programme doit avoir une fonction de démarrage appelée main qui indique le point d'entrée du programme.
4. Le début du corps de la fonction « main » et lui correspond dans l'algorithme « début »
5. Lieu des déclarations locales.
6. Instructions.
7. Retour: La fonction « main » doit renvoyer un entier (int). Il renvoie 0 au système d'exploitation, pour dire que tout s'est bien passé.
8. La fin de la fonction « main » et du programme, correspondant à celle-ci dans l'algorithme « fin ».

Observation

- La déclaration peut être faite indifféremment au lieu des déclarations publiques ou locales.
- La mise en forme du code, l'alignement, les marges, les espaces et le retour à la ligne après les caractères spéciaux tels que: ([{}]) =+, ;: etc. n'ont aucun sens dans le programme. La majeure partie du programme peut être écrite sur une seule ligne, ce qui sépare les instructions c'est ";" et pas de retour à la ligne.
- Le format d'écriture, l'alignement, les marges, les espaces et les retours à la ligne doivent être respectés pour la lisibilité du programme.

Un exemple montrant le processus de traduction d'un algorithme en C

algorithme	C	observation
`Algorithme surf_cercle`		Le nom de l'algorithme devient le nom du fichier surf_cercle.c
`Const P=3.14`	`Const float P=3.14;`	peut utiliser #define P 3.14
`Var r, s:entier`	`int r, s;`	Il n'y a pas de mot var en C

`//r le rayon et s surface`	`//r le rayon et s surface`	Cela ne fait pas partie du programme, mais seulement pour l'explication.
`Début`	`int main()` `{`	Les variables peuvent être déclarées après {
`Ecrire("Entrer le rayon")`	`printf("Entrez le rayon\n");`	\n pour revenir à la ligne
`Lire(r)`	`scanf("%d", &r);`	N'oubliez pas & avant la variable
`s←p*r*r`	`s=p*r*r;`	Chaque instruction se termine par ;
`Ecrire("L'aire du cercle est:" , s)`	`printf("L'aire du cercle est: %d" , s);`	Le format doit être donné
`Fin`	`}`	

exemple 2

Écrivez un programme qui calcule la moyenne pour ASD1.

```
#include <stdio.h>
int main() {
   float cont, td, tp, moy ;
   printf("Entrez la note d'examen\n") ;
   scanf("%f", &cont) ;
   printf("Entrez la note de TD\n") ;
   scanf("%f", &td) ;
   printf("Entrez la note de TP\n") ;
   scanf("%f", &tp) ;
   moy = (cont * 3 + td + tp) / 5 ;
   printf("La moyenne est %.2f" , moy) ;
   return 0;
}
```