INSTITUT SUPÉRIEUR D'ENSEIGNEMENT PROFESSIONNEL DE DIAMNIADIO
FILIÈRE TIC

ALGORITHME INTRODUCTION

2021-2022

Présenté par: **Abdoulaye MBAYE**

+221772389823

ambaye@isepdiamniadio.edu.sn

PLAN

- Introduction
- Quelques fondements de l'informatique
- Algorithmique Introduction

INTRODUCTION

- Pourquoi apprendre à programmer ?
  - Avez-vous besoin, durant vos études, de maîtriser les techniques fondamentales de programmation pour passer votre diplôme ?
  - Est-ce une nouvelle étape de votre carrière professionnelle où, n'étant pas informaticien, vous êtes amené à programmer des macros ou des scripts complexes ?
- Apprendre à programmer, c'est enfin savoir comment font les autres pour créer de superbes logiciels, c'est savoir à terme comment les créer soi-même, et les développer.

INTRODUCTION

- Comment apprendre à programmer ? On ne s'improvise pas programmeur. C'est un métier, et comme tout métier, cela s'apprend.

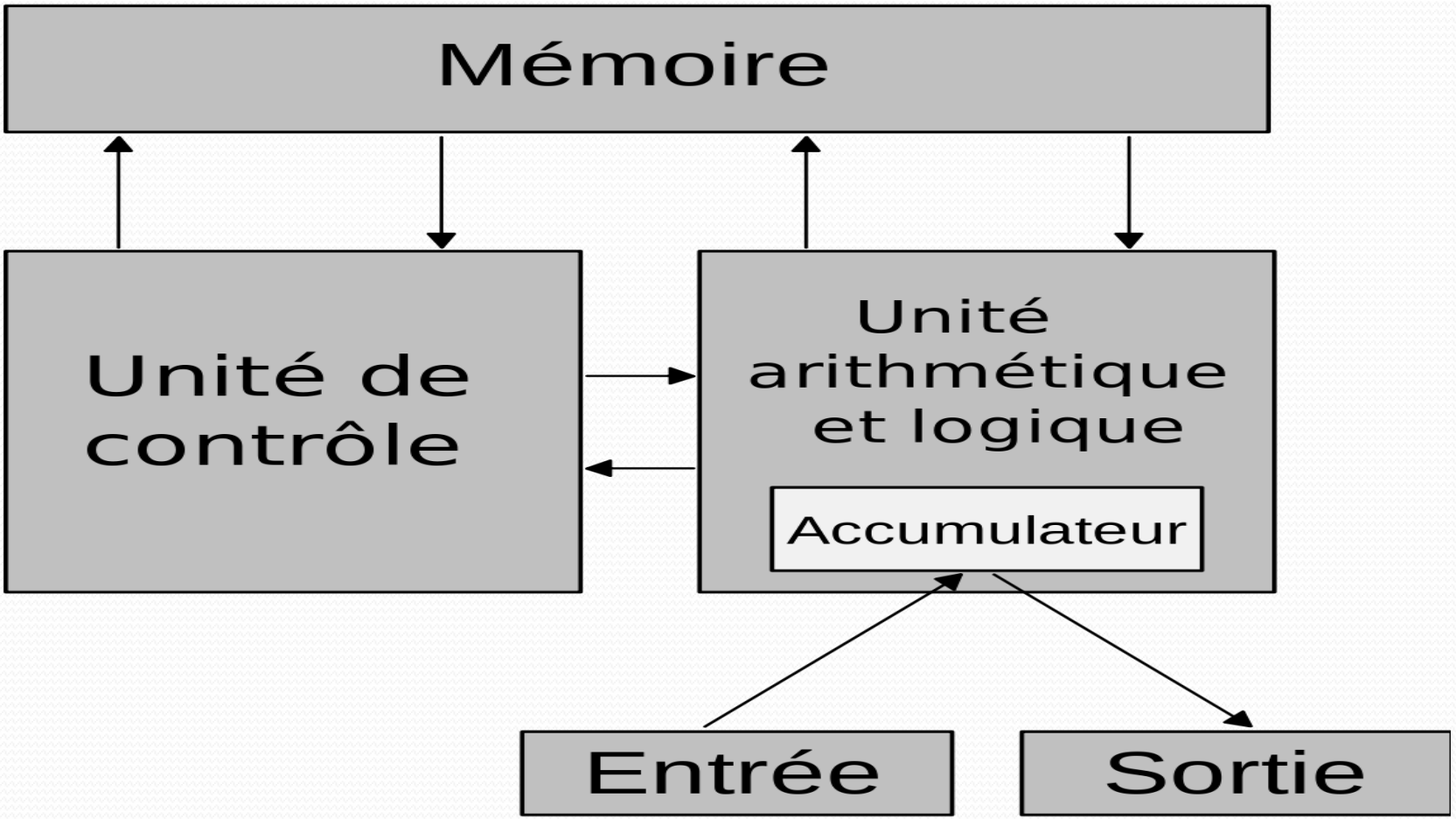
- **Tout le monde doit et va écrire des programmes**

  "La pratique rend parfait"

Les fondements de l'informatique

- **Architecture de Von Neumann**
  - L'architecture matérielle d'un ordinateur repose sur le modèle de Von Neumann qui distingue classiquement 4 parties:
    1. L'unité arithmétique et logique (UAL), ou unité de traitement, effectue les opérations de base (additions, soustractions, multiplications, divisions, modulos, gestion des signes (positif, négatif), opérations logiques (booléenne), comparaisons, parfois rotations et décalages de valeurs (toujours dans le cadre d'une logique booléenne)).
    2. L'unité de contrôle séquence des opérations.
    3. La mémoire contient à la fois les données et le programme qui dira à l'unité de contrôle quels calculs faire sur ces données. La mémoire se divise entre mémoire vive volatile (programmes et données en cours de fonctionnement) et mémoire de masse permanente (programmes et données de base de la machine).
    4. Les entrées-sorties sont des dispositifs permettant de communiquer avec le monde extérieur (écran, clavier, souris. . .).

Les fondements de l'informatique



Les fondements de l'informatique

- Les 2 premières parties sont souvent rassemblées au sein d'une même structure : le microprocesseur (la « puce »), unité centrale de l'ordinateur.

- **Machine de Turing**
  - C'est le mathématicien Anglais Alan Turing qui a trouvé et énoncé les bases théoriques de toute machine dite universelle, capable d'accomplir une série de tâches qu'on lui indique et dont les actions qu'elle réalise dépendent de ce qu'elle a déjà accompli.
  - Pour ce faire, Turing a notamment inventé la notion d'état (ou « étape »). Selon l'état dans laquelle elle se trouve, elle pourra exécuter des actions différentes face à une même situation.
  - Ce principe simple est le principe de base de tout processeur donc de tout système programmable.
  - Sans l'apport de Turing, il n'y aurait ni ordinateur ni téléphone portable ni tout autre dispositif contenant un processeur !

ALGORITHME: Introduction

- Un algorithme est une suite ordonnée d'instructions qui indique la démarche a suivre pour résoudre une série de problèmes équivalents.
  - Le premier algorithme fut crée par Al Khawarizmi vers l'an 820.
  - C'est son nom latinisé que vient le mot algorithme.
  - Le premier algorithme est une méthode qui a permis de trouver des solutions à un certain type d'équation.

ALGORITHME: Introduction

- Un algorithme est une suite d'actions ordonnées en séquence qui portent sur des objets. Ces objets doivent être définis de manière très précise.
- Un algorithme, c'est une suite d'instructions, qui une fois exécutée correctement, conduit à un résultat donné.
- Pour fonctionner, un algorithme doit donc contenir uniquement des instructions compréhensibles par celui qui devra l'exécuter.

ALGORITHME: Introduction

- Construction des algorithmes:
  - La construction des algorithmes informatiques suit une méthode appliquant un certain nombre de règles énumérées ci-dessous, règles qui forment les principes de la «Programmation Structurée ».

1. Méthode Descendante « Top-Down »

   Nous avons vu précédemment qu'une règle pour construire un algorithme est d'employer une méthode « Top-Down », donc de définir notre traitement en une décomposition successive de blocs. Nous ne reviendrons pas sur le principe de cette règle que nous appliquerons dans la suite de ce support de cours lorsque nous construirons des algorithmes informatiques complexes.

ALGORITHME: Introduction

2. Modularité

   Cette hiérarchisation devra respecter un autre principe « La Modularité », cela signifie que la décomposition ne devra pas mélanger des traitements de type différents.

3. Description des Données

   Toutes les données manipulées par l'algorithme devront être systématiquement décrites sous forme d'un pseudo code. Chaque donnée sera caractérisée par:

   - Un nom mnémonique
   - un type (numérique, caractère .etc….)
   - une longueur.

ALGORITHME:Introduction

- On distinguera :
  - les données en entrée : il s'agit des informations à traiter dans notre algorithme
  - les données en sortie : il s'agit des résultats que notre algorithme doit produire
  - les données de travail : ce sont des variables temporaires de stockage d'informations nécessaires à la réalisation de l'algorithme.
- En fonction de l'algorithme trouvé, le nombre de données de travail qui seront définies peut être variable. Il faut signaler, que la qualité d'un algorithme ne se mesure pas au nombre de données de stockage utilisées, mais à la lisibilité de l'information manipulée.

ALGORITHME: Introduction

4. Structures de Contrôle
   - Tout algorithme pourra être construit à partir de trois structures de contrôle
     - La séquence
     - L'alternative
     - L'Itérative ou la Répétitive
5. Représentation de l'Algorithme
   - Une fois défini, un algorithme sera décrit soit en utilisant :
   - Une représentation graphique « L'arbre L'Organigramme » Programmatique »,
   - Ou une représentation textuelle « Le Pseudo Code »

Il faut se rappeler que l'algorithme est indépendant du langage de programmation qui sera utilisé.

ALGORITHME: Introduction

- **Le pseudo-code(convention pour écrire de l'algorithme):** aussi appelé LDA (langage de description d'algorithme)
  - Est une description écrite étape par étape du code pour le transcrire progressivement dans la langue de code.
  - C'est à dire qu'une personne qui ne connaît pas le code ou ne le maîtrise pas forcement peut comprendre sans forcement coder.

ALGORITHME: Introduction

- Le pseudo-Code est avant tout écrit pour être compris par un humain et non par un ordinateur et doit rester compréhensible par tout les intervenants d'un projet, techniciens ou même si il est avant tout destiné à servir de ligne conductrice aux équipes de programme.

ALGORITHME: Introduction

- Exemple de **pseudocode**

```
Programme Nom

Variables
        nom1, nom2 : chaîne de caractère

Début
        Ecrire ("Donner votre nom, personne n°1" )
        Lire ( nom1 )
        Ecrire ("Donner votre nom, personne n°2" )
        Lire ( nom2 )
        Si ( Nom1 = Nom2 ) Alors
                Ecrire ("Vous avez le même nom!" )
                Sinon
                        Ecrire ("Vos noms sont différents" )
        Fin Si
Fin

Fin programme Nom
```

ALGORITHME: Introduction

- **Exemple:** Trouver son chemin Extrait d'un dialogue entre un touriste égare et un autochtone.
  - Pourriez-vous m'indiquer le chemin de la gare, s'il vous plait ?
  - Oui bien sur : vous allez tout droit jusqu'au prochain carrefour, vous prenez a gauche au carrefour et ensuite la troisième a droite, et vous verrez la gare juste en face de vous.
  - Merci.
- Dans ce dialogue, la réponse de l'autochtone est la description d'une suite ordonnée d'instructions (allez tout droit, prenez à gauche, prenez la troisième à droite) qui manipulent des données (carrefour, rues) pour réaliser la tache désirée (aller `a la gare). Ici encore, chacun a déjà été confronte à ce genre de situation et donc, consciemment ou non, a déjà construit un algorithme dans sa tête (ie. définir la suite d'instructions pour réaliser une tache). Mais quand on définit un algorithme, celui-ci ne doit contenir que des instructions compréhensibles par celui qui devra l'exécuter (des humains dans les 2 exemples précédents).

ALGORITHME: Introduction

- **validité d'un algorithme:** La validité d'un algorithme est son aptitude a réaliser exactement la tache pour laquelle il a été conçu.
  - Si l'on reprend l'exemple précédent de l'algorithme de recherche du chemin de la gare, l'étude de sa validité consiste à s'assurer qu'on arrive effectivement à la gare en exécutant scrupuleusement les instructions dans l'ordre annoncé
- **robustesse d'un algorithme:** La robustesse d'un algorithme est son aptitude à se protéger de conditions anormales d'utilisation.
  - Dans l'exemple précédent, la question de la robustesse de l'algorithme se pose par exemple si le chemin proposé a été pensé pour un piéton, alors que le « touriste égaré » est en voiture et que la « troisième à droite » est en sens interdit.

ALGORITHME: Introduction

- **réutilisabilité d'un algorithme:** La réutilisabilité d'un algorithme est son aptitude à être réutilisé pour résoudre des taches équivalentes à celle pour laquelle il a été conçu.
  - L'algorithme de recherche du chemin de la gare est-il réutilisable tel quel pour se rendre à la mairie ? A priori non, sauf si la mairie est juste à côté de la gare.
- **complexité d'un algorithme:** La complexité d'un algorithme est le nombre d'instructions élémentaires a exécuter pour réaliser la tache pour laquelle il a été conçu.
  - Si le « touriste égaré » est un piéton, la complexité de l'algorithme de recherche de chemin peut se compter en nombre de pas pour arriver `a la gare.

ALGORITHME: Introduction

- **efficacité d'un algorithme:** L'efficacité d'un algorithme est son aptitude à utiliser de manière optimale les ressources du matériel qui l'exécute.
  - N'existerait-il pas un raccourci piétonnier pour arriver plus vite à la gare ?
- L'algorithmique permet ainsi de passer d'un problème à résoudre à un algorithme qui décrit la démarche de résolution du problème. La programmation a alors pour rôle de traduire cet algorithme dans un langage « compréhensible » par l'ordinateur afin qu'il puisse exécuter l'algorithme automatiquement

ALGORITHME: Introduction

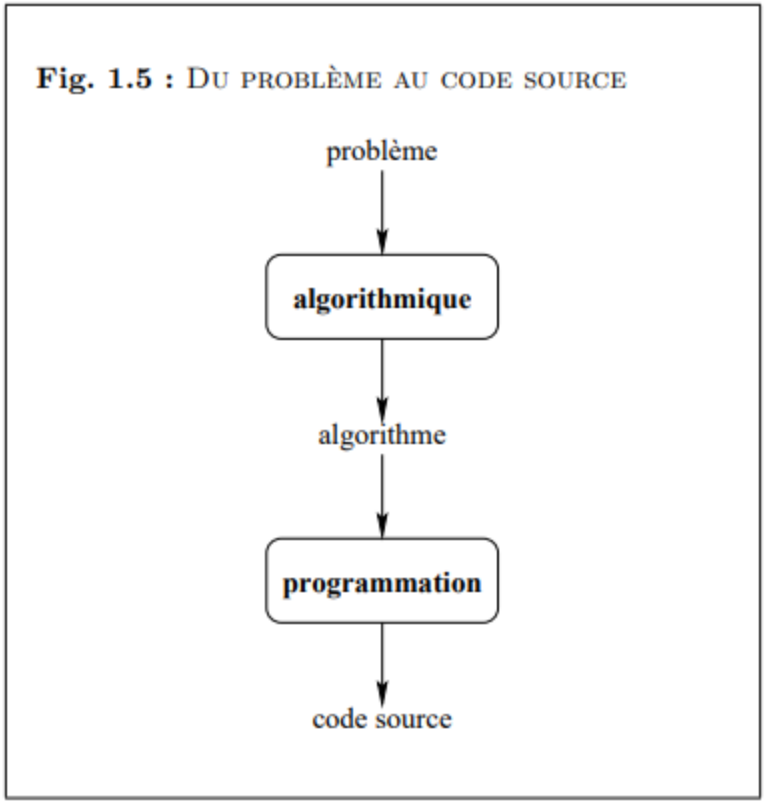


Fig. 1.5 : DU PROBLÈME AU CODE SOURCE

ALGORITHME: Introduction

- **<u>Exercice 1:</u> Dessins sur la plage :** exécution (1)
  - On cherche à faire dessiner une figure géométrique sur la plage à quelqu'un qui a les yeux bandés. Quelle figure géométrique dessine-t-on en exécutant la suite d'instructions ci-dessous ?
    - 1. avance de 10 pas,
    - 2. tourne à gauche d'un angle de 120° ,
    - 3. avance de 10 pas, 4. tourne à gauche d'un angle de 120° ,
    - 5. avance de 10 pas

ALGORITHME: Introduction

- **Exercice 2: Dessins sur la plage** : conception (1) Faire dessiner une spirale rectangulaire de 5 côtés, le plus petit côtés faisant 2 pas de long et chaque côté fait un pas de plus que le précédent.
- **Exercice 3: Propriétés d'un algorithme** Reprendre l' **exercice 1** et illustrer la validité, la robustesse, la réutilisabilité, la complexité et l'efficacité de l'algorithme propos´e pour dessiner sur la plage.

ALGORITHME: Introduction

- **Notion d'objet:**
  - Le traitement d'un objet concerne la valeur de cet objet.
  - Si cette valeur ne peut pas être modifiée, nous parlons de constante, sinon nous parlons de variable.
  - Un objet est parfaitement défini si nous connaissons ses trois caractéristiques, à savoir :
    - **Son identificateur** : il est représenté par une suite quelconque de caractères alphanumériques (numériques et alphabétiques sans espace) commençant obligatoirement par une lettre ou un tiret. De préférence, le nom est choisi en rapport avec le contenu de l'objet. Exemple: DELTA, Premiere_Racine .
    - **Sa valeur:** constante ou variable
    - **Son type** : nous ne pouvons pas appliquer de traitement à la valeur d'un objet si nous ne connaissons pas son type. Un type est défini par un ensemble de constantes et l'ensemble des opérations que nous pouvons leur appliquer.

ALGORITHME: Introduction

- **Il y a trois grands types d'objet :**
  - **Booléen**
    - constantes : vrai, faux
    - opérations : et logique, ou logique, non logique
  - **Numérique (entier ou réel)**
    - constantes : ensemble des entiers relatifs ou ensemble des réels
    - opérations : toutes les opérations arithmétiques et trigonométriques, notamment l'addition, la soustraction, la multiplication, la division, la puissance, la division entière, la partie entière d'un réel.
  - **Caractère :** c'est soit une lettre de l'alphabet latin, soit un code opération ('+', '-', '*', '/', '%', ...) ou un code de ponctuation (',', '.', ' :', ' !', ...) ou un code spécial ('&', '$', '#', ..).

Variables

- Qu'est ce qu'une variable ?
  - Un algorithme est fait pour résoudre un ensemble de problèmes semblables (cf définition du terme « algorithmique » dans le petit Robert). Par exemple, un logiciel qui gère la facturation d'une entreprise doit connaître les noms et adresses des clients, les produits commandés, etc. Ces informations ne peuvent pas être devinées par le programme et doivent donc être saisies par les utilisateurs. Ces informations sont appelées données en entrée. Elles proviennent de l'extérieur du programme et sont utilisées dans le programme.

Variables

- Inversement, le programme effectue des opérations, des calculs dont les résultats devront être transmis aux utilisateurs. Par exemple, le total d'une facture est calculée comme la somme de tous les produits commandés en multipliant le nombre d'articles par leur prix unitaire ; le total des ventes pour une période donnée est obtenu en faisant la somme des montants des factures, etc. Ces informations seront affichées à l'écran, imprimées, stockées dans des bases de données, etc. Ce sont des informations qui sortent du programme. On les appelle données en sortie.

Variables

- Le programmeur, pour décrire son algorithme utilise des variables pour représenter les données manipulées par un programme. Ce peut être les données en entrée, les données en sortie mais également les données résultant de calculs intermédiaires. Ainsi, pour calculer le montant d'une ligne d'une facture, le programmeur expliquera que le montant est obtenu en multipliant le prix unitaire par la quantité d'articles commandés. Il utilisera donc trois variables :
  - prix_unitaire, le prix unitaire de l'article ;
  - quantité, la quantité d'articles commandé ;
  - et montant, le montant de la ligne du bon de commande.

Variables: Définition

- Une variable peut être vue comme une zone dans la mémoire (vive) de l'ordinateur qui est utilisée pour conserver les données qui seront manipulées par l'ordinateur.
- Une variable est caractérisée par quatre informations :
  - son **rôle** : il indique à quoi va servir la variable dans le programme. Par exemple, on peut utiliser une variable pour conserver le prix unitaire d'un article, prix exprimé en euros. Cette information doit apparaître dans l'algorithme sous la forme d'un commentaire informel (texte libre) associé à la variable ;
  - son **nom** : composé uniquement de lettres minuscules, majuscules, de chiffres et du caractère souligné, il permet d'identifier la variable. On parle d'« identificateur ». Ce nom doit être significatif (refléter le rôle de la variable). Un compromis doit bien sûr être trouvé entre expressivité et longueur. On peut par exemple appeler prix_unitaire une variable qui représente le prix unitaire d'un article. Le nom prix_unitaire_d_un_article est un nom beaucoup trop long ;

Variables: Définition

- son **type** : une variable est utilisée pour représenter des données qui sont manipulées par le programme. Un type est utilisé pour caractériser l'ensemble des valeurs qu'une variable peut prendre. Par exemple le prix unitaire représente le prix d'un article exprimé en euros. On peut considérer qu'il est représenté par un réel. Il ne peut alors prendre que ce type de valeurs. De la même manière la quantité, ne peut prendre que des valeurs entières et le nom est une chaîne de caractères. On dira respectivement que le type de prix_unitaire est **Réel**, le type de quantité est **Entier** et le type de nom est **Chaîne**.
  - prix_unitaire: **Réel** //*prix unitaire d'un article (en euros)*
  - quantité: **Entier** // *quantité d'articles commandés*
  - nom: **Chaîne** //*nom de l'article*

Variables: Définition

- sa **valeur** : La variable contient une information qui peut varier au cours de l'exécution d'un programme. C'est cette information que l'on appelle valeur de la variable. La valeur d'une variable doit correspondre au type de la variable. Ainsi, une variable quantité de type entier pourra prendre successivement les valeurs de 10, 25 et 3.

- La valeur d'une variable n'existe que lorsque le programme est exécuté. Les autres informations (nom, rôle et type) sont définies lors de la conception du programme, pendant la construction de l'algorithme. Le rôle, le nom et le type sont des informations statiques qui doivent être précisées lors de la déclaration de la variable. En revanche, la valeur est une information dynamique qui changera au cours de l'exécution du programme

Variables: Définition

- **Règle** : Une variable doit toujours être initialisée avant d'être utilisée.
- **Définition** : Une variable est un nom désignant symboliquement un emplacement mémoire typé auquel est associé une valeur courante. Cette valeur peut être accédée ou modifiée.
- **Attention** : Le nom d'une variable doit être significatif : il doit suggérer, si possible sans ambiguïté, la donnée représentée par cette variable.
- **Règle** : En général, dès qu'on identifie une variable, il faut mettre un commentaire qui explique ce qu'elle représente et le rôle qu'elle joue dans l'algorithme.

TYPES FONDAMENTAUX

- **Définition** : 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.
- On appelle types fondamentaux les types qui sont définis dans notre langage algorithme par opposition aux types structurés (tableaux, enregistrements et types énumérés) qui doivent être définis par le programmeur.

TYPES FONDAMENTAUX

- **Intérêts** : 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 par 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.

TYPES FONDAMENTAUX

- **Opérateur** : À chaque type est associé un ensemble d'opérations (ou opérateurs).
- **Opérateurs de comparaison** : Tous les types présentés dans cette partie sont munis des opérations de comparaison suivantes : <, >, <=, >=, = et <> ou ≠.
- Les opérateurs de comparaison sont des opérateurs à valeur booléenne.
- **Attention** : Tous les types manipulés par une machine sont discrets et bornés. Ces limites dépendent du langage de programmation et/ou de la machine cible considérés. Aussi, nous n'en tiendrons pas compte ici. Il faut cependant garder à l'esprit que toute entité manipulée est nécessairement bornée.

TYPES FONDAMENTAUX

- **Les entiers:** Le type **Entier** caractérise les entiers relatifs.
  - **Exemple: 0; 12; 243; 10;** //*Exemple de représentants des entiers*

- Les opérations sont : **+, -, *, Div** (division entière), **Mod** (reste de la division entière) et **Abs** (valeur absolue) - et + peuvent être unaires ou binaires.
  - 10 **Mod** 3 // -- *1 (le reste de la division entière de 10 par 3)*
  - 10 **Div** 3 //-- *3 (le quotient de la division entière de 10 par 3)*
  - 1 **Div** 2 //-- 0 *(le quotient de la division entière de 1 par 2)*
  - **Abs**(-5) //-- *5 (l'entier est mis entre parenthèses)*

TYPES FONDAMENTAUX

- Les **réels**: Le type **Réel** caractérise les réels.
  - // -- *Exemple de représentants des réels*
  - 10.0
  - 0.0
  - -10e-4
- Les opérations sont : +, -, *, /, **Abs** (valeur absolue), Trunc (partie entière). - et + peuvent être unaires ou binaires.

TYPES FONDAMENTAUX

- Les **booléens:** le type **Booléen** caractérise les valeurs booléennes qui ne correspondent qu'à deux valeurs **VRAI** ou **FAUX**.
  - Les opérations sont **Et**, **Ou** et **Non** qui sont définies par la table de vérité suivante :

A	B	A **Et** B	A **Ou** B	**Non** A
VRAI	VRAI	VRAI	VRAI	FAUX
VRAI	FAUX	FAUX	VRAI	FAUX
FAUX	VRAI	FAUX	VRAI	VRAI
FAUX	FAUX	FAUX	FAUX	VRAI

TYPES FONDAMENTAUX

- Lois de De Morgan : Les lois de De Morgan sont particulièrement importantes à connaître.
  - **Non** (A **Et** B) = (**Non** A) **Ou** (**Non** B)
  - **Non** (A **Ou** B) = (**Non** A) **Et** (**Non** B)
  - **Non** (**Non** A) = A

TYPES FONDAMENTAUX

- Les **caractères**: Le type **Caractère** caractérise les caractères.
  - */////Exemple de représentants des caractères*
  - ‘a’ *//le caractère a*
  - ‘(’ *//le caractère (*
  - ‘=’ *//le caractère =*
- On considère que les lettres majuscules, les lettres minuscules et les chiffres se suivent. Ainsi, pour savoir si un variable c de type caractère correspond à un chiffre, il suffit d’évaluer l’expression (c >= ’0’) Et (c <= ’9’).
- **Opérations** :
  - **Ord** : Permet de convertir un caractère en entier.
  - **Chr** : Permet de convertir un entier en caractère.
  - Pour tout c: **Caractère** on a Chr(Ord(c)) = c.

TYPES FONDAMENTAUX

- Les **chaînes de caractères:** Le type **Chaîne** caractérise les chaînes de caractères.
  - "Une chaîne de caractères" // *un exemple de Chaîne*
  - "Une chaîne avec guillement (\")" // *un exemple de Chaîne*
- **Attention** : Il ne faut pas confondre le nom d'une variable et une constante chaîne de caractères !

CONSTANTES

- Certaines informations manipulées par un programme ne changent jamais. C'est par exemple la cas de la valeur de π , du nombre maximum d'étudiants dans une promotion, etc.
- Ces données ne sont donc pas variables mais constantes. Plutôt que de mettre explicitement leur valeur dans le texte du programme (constantes littérales), il est préférable de leur donner un nom symbolique (et significatif). On parle alors de constantes (symboliques).

CONSTANTES

- PI = 3.1415 // *Valeur de PI*
- MAJORITÉ = 18 //*Âge correspondant à la majorité*
- TVA = 19.6 //*Taux de TVA en vigueur au 15/09/2000 (en %)*
- CAPACITÉ = 120 //*Nombre maximum d'étudiants dans une promotion*
- INTITULÉ = "Algorithmique et programmation" //*par exemple*

CONSTANTES

- **π** peut être considérée comme une constante absolue. Son utilisation permet essentiellement de gagner en clarté et lisibilité. Les constantes MAJORITÉ, TVA, CAPACITÉ, etc. sont utilisées pour paramétrer le programme.
- Ces quantités ne peuvent pas changer au cours de l'exécution d'un programme. Toutefois, si un changement doit être réalisé (par exemple, la précision de PI utilisée n'est pas suffisante), il suffit de changer la valeur de la constante symbolique dans sa définition (et de recompiler le programme) pour que la modification soit prise en compte dans le reste de l'algorithme.

LES EXPRESSIONS

- Une expression est « quelque chose » qui a une valeur. Ainsi, en fonction de ce qu'on a vu jusqu'à présent, une expression est une constante, une variable ou toute combinaison d'expressions en utilisant les opérateurs arithmétiques, les opérateurs de comparaison ou les opérateurs logiques.
- ***Une expression*** peut être:
  - Soit une constante. Ex: 27, 'Algérie', faux.
  - Soit une variable. Ex: B, A13, Nom.
  - Soit appel de fonction. Ex: sin (x), abs (y),…
  - Soit une opération simple. Ex: -x*y.
  - Soit une expression complexe qui contient plusieurs opérateurs. Ex: a + b/5 – abs(y) * 3.
  - rayon // *une variable de type réel*
  - 2*rayon //*l'opérateur * appliqué sur 2 et rayon*
  - 2*PI*rayon // *une expression mêlant opérateurs, constantes et variables*
  - rayon >= 0 // *expression avec un opérateur de comparaison*

- **Attention** : Pour qu'une expression soit acceptable, il est nécessaire que les types des opérandes d'un opérateur soient compatibles. Par exemple, faire l'addition d'un entier et d'un booléen n'a pas de sens. De même, on ne peut appliquer les opérateurs logiques que sur des expressions booléennes.
- Quelques règles sur la compatibilité :
  - Deux types égaux sont compatibles.
  - On peut ensuite prendre comme règle : le type A est compatible avec le type B si et seulement si le passage d'un type A à un type B se fait sans perte d'information. En d'autres termes, tout élément du type A a un correspondant dans le type B. Par exemple, **Entier** est compatible avec **Réel** mais l'inverse est faux.

- Si l'on écrit n + x avec n entier et x réel, alors il y a « convertion » : l'entier n est converti en un réel puis l'addition est réalisée sur les réels, le résultat étant bien sûr réel.
- **Règle d'évaluation d'une expression :** Une expression est évaluée en fonction de la priorité des opérateurs qui la composent, en commençant par ceux de priorité plus forte. À priorité égale, les opérateurs sont évalués de gauche à droite.

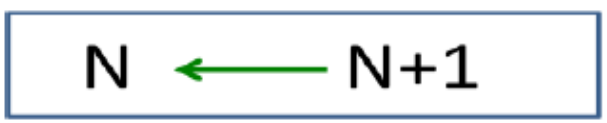
- Voici les opérateurs rangés par priorité décroissante 3 :
  - . (accès à un champ) la priorité la plus forte
  - +, -, **Non** (unaires)
  - *, /, **Div**, **Mod**, **Et**
  - +, -, **Ou**
  - <, >, <=, >=, = et <> priorité la plus faible
- Il est toujours possible de mettre des parenthèses pour modifier les priorité.
  - prix_ht * (1 + tva) // *prix_ttc*
  - En cas d'ambiguïté (ou de doute), il est préférable de mettre des parenthèses.

Les instructions élémentaires

- Se sont des opérations primitives (traitements) sur les objets:
  - ***Affectation:*** Permet d'affecter une valeur à une variable.
  - Sa forme: <identificateur> ← <expression>;
  - **Exemple:** Y ← 3; x ← 5; x1 ← y + x; y1 ← x1 + 2; y1 ← y1*2;
- **Remarque:**
  - La valeur affectée doit être de même type que la variable.
  - Le contenu de la variable est alors écrasé et son nouveau contenu sera la valeur affectée.

Les instructions élémentaires

- ***Incrémentation et décrémentation***



Incrémentation de 1 (augmentation)

Décrémentation de 1 (diminuer)