Programmation

Algorithmie : Module 1 Partie 1

Objectifs :

* Organiser un programme
* Aborder la programmation, présentation d’un langage universelle
* Utiliser des données élémentaires
* Structurer les étapes de résolution d’un problème

Prérequis :

Histoire et modèle de Von Neumann

**Algorithmique : Suite finie de règles à appliquer dans un ordre déterminé a un nombre fini de données pour arriver, en nombre finie d’étapes à un certain résultat et cela indépendamment des données**

Nécessité :

* Les langages de hauts niveaux évoluent très vite, l’algorithme a traversé 40 ans de langages de haut niveau tout en s’adaptant aux nouveautés.
* L’écriture algorithmiques permet de prendre de la distance face aux spécificités des langages évolué pour lesquels la compréhension ainsi que la construction de nouveaux programmes se sont considérablement alourdies
* La validation des structures algorithmiques est bien plus aisée que la validation des langages évolués
* Le travail en équipe nécessite un langage commun, puis en répartition suivant le langage évolué approprié aux spécificités du moment

Procédure de réalisation d’un programme informatique :

* L’écriture algorithmique est une phase intermédiaire et indispensable pour réaliser un programme.
* La qualité du développement final dépend aussi de cette phase cruciale

Nom\_de\_la\_vaiable : < TYPE > note\_sur\_20 : Entier

Nom attribué à la variable déclarée le type est toujours précède du signe **:**

Les types élémentaires possibles sont :

* Les entiers n\_nombre : Entier
* Les réels
* Les caractères c\_caractère : Caractère
* Les chaînes de caractères s\_chaine : chaine
* Les booléens

Nom\_var1,var2,var3 : < TYPE >

Nom attribué aux variables déclarées le type est toujours précède du signe **:**

Une variable ne peut avoir qu’un seul et unique type

Affectation : (nom de la variable)

Nom\_de\_la\_variable valeur note\_sur\_20 15 (valeur)

(Symbole d’affectation)

On peut affecter à une variable la valeur d’une autre variable :

Nom\_de\_la\_variable nom\_variable2

Note\_sur\_100 note\_sur\_20 \* 5

Réaffectation :

Note\_sur\_100 note\_sur\_100 + 5

**Les constantes :**

Les variables : VAR

* Déclaration
* Affectation
* Utilisation
* Réaffectation

Les constantes : CONST

* Initialisation
* Utilisation

La valeur d’une constante reste identique tout ou long du programme

Nom\_de\_la\_constante valeur : < Type >

Pi 3.14 : REEL

Pourquoi utiliser une constante plutôt qu’une variable ?

* Optimisation
* Fiabilité
* Souplesse
* Facilité de relecture

Les données manipulées par les instructions sont :

* Des variables
* Des constantes
* Des valeurs littérales (VRAI, 45)
* Des expression complexes (combinaison de variables, constantes et valeurs littérales avec des opérateurs)

Primitives :

* Une institution est un ordre élémentaire au sens algorithmique, que peut exécuter un programme

Deux instructions d’entrée-sortie :

* LIRE : lecture de la frappe (au clavier)
* ECRIRE : affichage (écran)
* Début
* ACTION 1 LIRE valeur
* ACTION 2 carre valeur \* valeur
* ACTION 3 ECRIRE carre
* FIN

Structure d’un algorithme :

* Nom de l’algorithme
* Spécifications de l’algorithme
* Déclarations
* Corps de l’algorithme

Conventions d’écritures :

Les commentaires :

* Sont précédés de : //

Les mots clés et primitives :

* Sont écrits en majuscule : VAR :

Les chaînes de caractères sont encadrées par :  « »

Les caractères sont encadrés par : ‘ ’

Les accents sont seulement autorisés :

* Dans les lignes de commentaires
* Dans les chaînes de caractères : « élève »
* Pour les caractères uniques : ‘à’
* Début
* ACTION 1 LIRE valeur
* ACTION 2 valeur valeur \* valeur
* ACTION 3 ECRIRE valeur
* Fin

Valeurs intermédiaires :

On ne connait plus le valeur initiale entrée par l’utilisateur

Il serait plus « user friendly » d’utiliser l’instruction

* ECRIRE « le carré de »
* ECRIRE valeur
* ECRIRE « est »
* ECRIRE produit

**Structures :**

* Structures algorithmiques :
  + Séquences
  + Ruptures de séquences
* Structure de données :
  + Structures élémentaires
  + Structures avancées
* Structures de programmes

**Les types élémentaires possible sont :**

* Les entiers
* Les réels
* Les caractères
* Les chaines de caractères
* Les booléens

*ENTIER*

* Nombres entiers
* Négatifs
* Positifs

Exemples :

* 33
* -13

Opérations possibles :

* Addition
* Soustraction
* Multiplication
* Division
* Exposant
* Modulo
* Comparaison

*REEL*

* Nombres entiers
* Fractionné
* Négatifs
* Positifs
* Nuls

Opérations possibles :

* Addition
* Soustraction
* Multiplication
* Division
* Exposant
* Comparaisons

*CARACTERES : CAR*

Caractères uniques

Exemples :

* ‘F’
* ‘I’
* ‘ !’

Opérations possibles Opérateurs associés, symboles, mot clés correspondants

* + Comparaison < <= > >= = <>
  + Extractions car1 Extraction (chaine, position)

CHAÎNE

* Caractère unique
* Suite de caractères

|  |  |
| --- | --- |
| Opérations possibles | Opérateurs associés, symbole, mots clés correspondants |
| Comparaison | < <= > >= = <> |
| Extraction | Ch1 Extraction (ch2, position, longueur) |
| Longueur | Ent Longueur (ch) |
| Concaténation | Ch Concatener (ch1, ch2) ou ch ch1 + ch2 |

BOOLEEN

|  |  |
| --- | --- |
| Opérations possibles | Opérateurs associées |
| Comparaison | = <> |
| Négations | NON |
| Conjonction | ET |
| Disjonction | OU |

VAR

* Mot,motmirroir : chaine
* Motegal : booleen

Début

* + - Ecrire « veuillez entrer votre mot »
    - LIRE mot
    - Mot EXTRACTION (mot, ‘’\_\_\_\_\_’’,1,5)
    - Motmirroir EXTRACTION (mot,5,1) + EXTRACTION (mot,4,1) + EXTRACTION (mot,3,1) + EXTRACTION (mot,2,1) + EXTRACTION (mot,1,1)
    - Motegal (motmirroir=mot)
    - ECRIRE motegal

Fin

**Notions de rupture de séquences :**

L’exécution des opérations élémentaires (ou ACTIONS 2a et 2b) n’est pas systématique. Elle dépend d’une condition

Catégories :

* Séquences
* Rupture de séquences :
  + Rupture conditionnelle (et non répétitive)
    - SI
    - CAS
  + Rupture répétitive
    - POUR
    - TANT QUE
    - REPETER

Rupture conditionnelle :

* SI…ALORS…FINSI
* SI…ALORS…SINON…FINSI
* CAS…PARMI…
* CAS…PARMI…PARDEFAUT…
* SI <condition> ALORS
* <Instructions>
* [SINON
* <Instructions>
* ]
* FINSI
* CAS <variable> PAMRI
* {CAS {value [,]} :<instructions>}
  + [PARDEFAUT : <instructions>]
  + FINCASPARMI

Ruptures de répétition :

* POUR…FAIRE…FINPOUR
* TANTQUE…FAIRE…FINTANTQUE
* REPETER…JUSQU’A…
* REPETER…TANTQUE…

POUR <variable entière> DE < variable ou constante entière> A <variable ou constante entière> [PAS <variable ou constante entière>]

FAIRE

* [<instructions>]

FINPOUR

Mauvais algo : à ne pas faire

DEBUT

* LIRE nombre
* POUR compteur DE nombre A 1 pas -1
* FAIRE
  + ECRIRE « valeur du compteur à rebours » + compteur
* FINPOUR

FIN

Bon algo : à faire

VAR :

* Nombre, compteur : ENTIER

DEBUT

* LIRE nombre
* POUR compteur DE 0 A nombre -1
* FAIRE
  + ECRIRE « Valeurs du compteur à rebours : » + nombre – compteur
* FINPOUR

FIN

TANTQUE

TANTQUE <exp booléenne> FAIRE

{<Instructions>}

FINTANTQUE

Algo : Compteurs rebours

VAR :

* Nombre : ENTIER

DEBUT

* LIRE (nombre)
* TANTQUE (nombre >0)
* FAIRE
  + ECRIRE (« valeur du compteur à rebours : » + nombre)
  + nombre nombre – 1
* FINTANTQUE

FIN

**REPETER JUSQU’A**

Algo : compteurs rebours

VAR :

* Nombre : ENTIER

DEBUT

* LIRE (nombre)
* REPETER
  + ECRIRE (« Valeur du compteur à rebours : » + nombre)
  + Nombre <- nombre – 1
* JUSQU’A (nombre <0)

FIN

Algo : compteur\_rebours3

VAR :

* Nombre : ENTIER

DEBUT

* LIRE (nombre)
* REPETER
  + ECRIRE (« Valeurs du compteur à rebours)
  + Nombre 🡨 nombre – 1
* TANTQUE (nombre > 0)

FIN