Les algorithmes – Bloc C

Objectifs – 21 heures de travail (avec laboratoires)

Habiletés à développer:

- Identifier les types de variables ;
- Énoncer les principales étapes à suivre ;
- Identifier les entrées et les sorties ;
- Traiter des données ;

Questions et compréhension

- Êtes-vous à l'aise avec les algorigrammes ?
- Êtes-vous en mesure de comprendre le *flow* de traitement ?

La compétence de la compréhension du *flow* de traitement se développe avec le temps et la pratique. Assurez-vous de comprendre la globalité et le principe général du traitement.

La compréhension du *flow* de traitement est un prérequis au développement du programmeur. Assurez-vous de développer cette compétence! Quelle est la solution pour développer la compétence ? La pratique !

Les algorithmes et le pseudo-code

Définition, explication et compréhension

Le pseudo-code

Qu’est-ce que le pseudo-code ?

Définition Wikipédia:

« *En programmation, le **pseudo-code**, également appelé **LDA** (pour **Langage de Description d'Algorithmes**) est une façon de décrire un algorithme en langage presque naturel, sans référence à un langage de programmation en particulier.*

L'écriture en pseudo-code permet souvent de bien prendre toute la mesure de la difficulté de la mise en œuvre de l'algorithme, et de développer une démarche structurée dans la construction de celui-ci. En effet, son aspect descriptif permet de décrire avec plus ou moins de détail l'algorithme, permettant de ce fait de commencer par une vision très large et de passer outre temporairement certains aspects complexes, ce que n'offre pas la programmation directe. »

source : https://fr.wikipedia.org/wiki/Pseudo-code

Formalisme des algorithmes et pseudo-code

Règles d'un algorithme :

- Contient un entête
    - Nom : le nom de l'algorithme ;
    - Rôle : ce que l'algorithme fait ;
    - Données : les données en entrées ;
    - Résultat : ce que l'algorithme produit en sortie ;
- Contient un corps
    - Il est délimité par les mots clés début et fin ;
    - Contient un lexique expliquant les variables ;
- Les identifiants des variables sont représentatifs de la donnée;
- Le nom de la fonction doit être relié à son traitement.

Formalisme des algorithmes et pseudo-code

Exemple d'algorithme:

Nom : AdditionnerDeuxEntiers

Rôle : Additionner deux entiers et mémoriser le résultat

Données : les valeurs à additionner

Résultat : la somme des deux valeurs

```
DÉBUT
                C <- a + b
FIN
```

Lexique:

a : entier

b : entier

c : entier

Les variables

Les variables sont des entités qui contiennent une information. Elles possèdent:

- Un nom (on parle de son identifiant qui doit être UNIQUE)
- Une valeur
- Un type qui caractérise l'ensemble des valeurs que peut prendre la variable (entier, décimal, chaine de caractère, etc.)

L'ensemble des variables est virtuellement stocké dans la mémoire de l'ordinateur.

Exemple de types de variables

Type	Description
Entier	Pour manipuler des entiers
Réel	Pour manipuler des nombres réels (les **nombres réels** sont des nombres qui peuvent être représentés par une partie entière et une liste finie ou infinie de nombre décimaux)
Booléen	Pour les valeurs booléenne
Caractère	Pour manipuler des caractères alphabétiques
Chaîne	Pour manipuler des chaînes de caractères permettant de représenter des mots ou des phrases

Association de données à une variable

Les variables permettent de conserver virtuellement une information dans la mémoire de l'ordinateur.

Elles doivent donc être associées à une valeur!

Exemple: c <- 45

La valeur 45 est associée à la variable c

Opérande, opérateur et expression

- Un opérande est une entité (variable, constante ou expression) utilisée par un opérateur
- Un opérateur est un symbole d'opération qui agit sur des variables ou qui permet de faire des calculs (+, -, /, %, etc.)
- Une expression est une combinaison d'opérateurs et d'opérandes.

Exemple d’opérande, d’opérateur et expression

Explication de l’expression:

A + B

- A est l’opérande de gauche ;
- + est l’opérateur ;
- B est l’opérande de droite ;
- A + B est une expression ;

Si A vaut 3 et que B vaut 4 alors l’expression A + B vaut 7.

Les opérateurs

Les opérateurs ne peuvent pas additionner un entier et un caractère mais il peut changer en fonction du type d'opérande.

Exemple: 2 + 5 vaut 7

« Bonjour » + « tout le monde » vaut « Bonjour tout le monde ».

L'opérateur s'ajuste en fonctions des opérandes

Les opérateurs booléens

Les opérateurs booléens:

- Valeurs possibles = Vrai ou Faux
- Associativité des opérateurs et
  - a et (b et c) = (a et b) et c
- Commutativité des opérateurs et/ou
  - a et b = b et a
  - a ou b = b ou a

Les valeurs booléennes PEUVENT être associatives et commutatives. Tout dépend des priorités (à voir plus loin)

Les opérateurs booléens

Les opérateurs booléens permettent d'avoir un résultat binaire.

Négation

a	non a
Vrai	Faux
Faux	Vrai

Et

a	b	a et b
Vrai	Vrai	Vrai
Vrai	Faux	Faux
Faux	Vrai	Faux
Faux	Faux	Faux

Les opérateurs booléens

Les opérateurs booléens permettent d'avoir un résultat binaire.

Ou

a	b	a ou b
Vrai	Vrai	Vrai
Vrai	Faux	Vrai
Faux	Vrai	Vrai
Faux	Faux	Faux

Ou exclusif

a	b	a ouExclusif b
Vrai	Vrai	Faux
Vrai	Faux	Vrai
Faux	Vrai	Vrai
Faux	Faux	Faux

Les opérateurs numériques

On retrouve naturellement les signes +, -, *, /

On y additionne l'élément *modulo(%)* que nous utilisons régulièrement.

Exemple de *modulo : 11 mod 2 = 1* mais 10 mod 2 = 0

Les opérateurs sur les numériques

L’opérateur d’égalité (=) permet de savoir si deux opérandes sont égales. On utilise le signe pour pouvoir effectuer cette vérification.

L’opérateur d’inégalité (!=) permet de valider que deux opérandes sont inégales.

Les opérateurs <, >, <= et >= permettent de vérifier des ordres de grandeur.

Priorités des opérateurs

Tout comme en arithmétique, les opérateurs ont également des priorités. Par exemple, les * et les / sont prioritaires sur les + et les -.

Pour les booléens, la priorité des opérateurs est la ***négation***, le ***et***, le ***ou exclusif*** et le ***ou***.

Pour clarifier les ambiguïtés lorsqu'il peut y en avoir, il faut utiliser les parenthèses.

Manipulation des variables

On peut utiliser les variables pour deux actions:

- Obtenir son contenu
  - Cette action d'obtenir son contenu se fait uniquement en utilisant le nom de la variable
- Affecter un nouveau contenu
  - Cette action s'effectue en affectant une nouvelle valeur à la variable et elle s'effectue en utilisant le symbole <- ou 
  - Exemple : identifiant de la variable <- expression

Manipulation de variables

Exemple: c <- a + b s'interprète de la façon suivante :

- On prend la valeur contenu dans la variable a ;
- On prend la valeur contenu dans la variable b ;
- On additionne les deux valeurs ;
- On affecte le résultat dans la variable c ;

Si la variable c avait auparavant une valeur alors cette dernière est perdue!

Les entrés et les sorties

- Un algorithme peut avoir des interactions avec l'utilisateur ;
- Il peut afficher un résultat ;
- Il peut demander à l'utilisateur de saisir une information afin de pouvoir la stocker dans une variable ;

Pour écrire/afficher on utilise:

- Afficher/Écrire « Bonjour et bienvenue à la formation »

Ces informations permettent de simuler un affichage à un écran

Les entrés et les sorties

- Les instructions de lecture permettent le prise de données
- Exemple: variable <- lire()
- Note: Les fonctions nécessitent, en général, des parenthèses.

- L’exécution de cette instruction consiste à affecter une valeur à la variable.

Exemple de pseudo-code

On veut écrire un algorithme qui lit une valeur représentant un montant d'argent. Ensuite, il additionne les éléments entrés.

- Nom : CalculerSomme.
- Rôle : Effectuer la somme
- Données : la somme des montants.
- Résultat : Le total.

Principe : on commence par lire sur l'entrée une série de 4 nombres. Ensuite, il faut additionner les nombres ensemble et retourner le total.

Suite pseudo-code

```
début
        nombre1 <- lire()
        nombre2 <- lire()
        nombre3 <- lire()
        nombre4 <- lire()
        Somme <- nombre1 + nombre2 + nombre3 + nombre4
        Afficher «  La somme est de » Somme
fin
```

Les structures de contrôles

Les structures de contrôle sont les éléments vous permettant de contrôler le '*flow*' de votre application. C'est grâce aux structures de contrôle que les logiciels ne partent pas en vrille dans un monde virtuel complètement dénué de sens.

Il y a trois (3) structures de contrôles principales:

1. Les structures conditionnelles (si et sinon)
2. Les boucles
3. Les fonctions

Les branchements

Pour qu'un algorithme puisse s'exécuter correctement il faut être en mesure d'effectuer des branchements et de choisir ce qui doit être exécuté.

Les instructions 'si alors' et 'sinon' permettent de conditionner les exécutions d'un algorithme à une valeur booléenne.

Syntaxe :

si *expression booléenne* **alors**

suite d'instructions exécutées si l'expression est vraie

sinon

suite d'instructions exécutées si l'expression est fausse

finsi

Les branchements

La partie du ‘sinon’ est optionnelle, on peut donc avoir la syntaxe suivante :

si *expression booléenne* **alors**

suite d’instructions exécutées si l’expression est vraie

finsi

Les branchements

Exemple d'un branchement:

```
début
        si valeur  ≥ 0 alors
                Effectuer un traitement X
        sinon
                Effectuer un traitement Y
        finsi
fin
```

Portez une attention à l'endroit où se ferme le 'finsi'.

Plusieurs si

Lorsque l'on doit comparer une **même** variable avec plusieurs valeurs, comme par exemple :

```
si a=1 alors
        faire une chose
    sinon
        si a=2 alors
            faire une autre chose
        sinon
            si a=4 alors
                faire une autre chose
            sinon
            . . .
            finsi
        finsi
    finsi
```

Il faut éviter d'avoir trop d'imbrications. Il faut plutôt utiliser le mode 'cas'.

Utilisation du ‘cas’

Sa syntaxe est :

cas où v **vaut**
v1 : action1
v2 : action2
. . .
vn : action
autre : action autre
fincas

Ainsi, le code est plus propre et beaucoup plus lisible

Les itérations

Il arrive souvent dans un algorithme qu'une même action soit répétée plusieurs fois, avec éventuellement quelques variantes. Il est alors fastidieux d'écrire un algorithme qui contient de nombreuses fois la même instruction. De plus, ce nombre peut dépendre du déroulement de l'algorithme.

Il est alors impossible de savoir à l'avance combien de fois la même instruction doit être décrite.

Pour gérer ces cas, on fait appel à des instructions en boucle qui ont pour effet de répéter plusieurs fois une même instruction.

Deux formes existent :

La première, si le nombre de répétitions est connu avant l'exécution de l'instruction de répétition;

La seconde s'il n'est pas connu. L'exécution de la liste des instructions se nomme itération.

La boucle ‘pour’

Il est fréquent que le nombre de répétitions soit connu à l'avance, et que l'on ait besoin d'utiliser le numéro de l'itération afin d'effectuer des calculs ou des tests. Le mécanisme permettant cela est la boucle Pour.

Forme de la boucle Pour :

```
Pour variable de valeur initiale à valeur finale faire
        liste d'instructions
finpour
```

Pour les itérations inconditionnelles

La boucle ‘tantque’

L'utilisation d'une "boucle pour" demande de connaître à l'avance le nombre d'itérations désiré, c'est-à-dire la valeur finale du compteur. Dans beaucoup de cas, on souhaite répéter une instruction tant qu'une certaine condition est remplie, alors qu'il est a priori impossible de savoir à l'avance au bout de combien d'itérations cette condition cessera d'être satisfaite. Dans ce cas, on a deux possibilités :

la boucle Tant que

la boucle Répéter jusqu'à

Syntaxe de la boucle Tant que :

```
tant que condition faire
    liste d'instructions
ftant
```

Boucle ‘tant que’ vérification à la fin

Parfois, il est nécessaire d’effectuer une fois la boucle et de vérifier à la fin l’état de la condition.

Syntaxe:

```
Faire
        liste d'instructions
Tant que condition faire
```

Les tableaux

Lorsque les données sont nombreuses et de même type, afin d'éviter de multiplier le nombre des variables, on les regroupe dans un tableau. Ainsi, le type d'un tableau précise le type (commun) de tous les éléments.

Syntaxe :

tableau type_des_éléments[borne_inf ... borne_sup]

La valeur 0 pour la borne inférieure dans le but de faciliter la traduction de l'algorithme vers les autres langages (C, Java, C#). Pour un tableau de 10 entiers, on aura :

tab : tableau entier[0..9]

Les tableaux

Ce tableau est de longueur 10, car il contient 10 emplacements. Le premier élément se situe à la case zéro.

12	20	458	24	12	7	578	125	222	666

Chacun des dix nombres du tableau est repéré par son rang, appelé indice. Pour accéder à un élément du tableau, il suffit de préciser entre crochets l'indice de la case contenant cet élément.

Par exemple, pour accéder au 5ème élément (12), on écrit : Tab[4]

Important de saisir le concept

Les tableaux

Les instructions de lecture, écriture et affectation s'appliquent aux tableaux comme aux variables.

x ← Tab[0]

La variable x prend la valeur du premier élément du tableau, c'est à dire : 12

12	20	458	24	12	7	578	125	222	666

Tab[6] ← 578

Cette instruction a modifié le contenu du tableau

Les tableaux

La plupart des algorithmes basés sur les tableaux utilisent des itérations permettant de faire un parcours complet ou partiel des différents éléments du tableau. De tels algorithmes établissent le résultat recherché par récurrence en fonction des éléments successivement rencontrés.

Les répétitions inconditionnelles sont le moyen le plus simple de parcourir complètement un tableau.

Les tableaux

Dans l'exemple suivant, le programme initialise un à un tous les éléments d'un tableau de n éléments :

```
        InitTableau
début
    pour i de 0 à n-1 faire
        tab[i] ← 0
    fpour
fin
```

Les fonctions

Les fonctions permettent de faire un traitement sur des données.

Par exemple:

```
Début
        TraiterLesNombres()
Fin

Fonction TraiterLesNombres()
        Instructions à effectuer
Fin fonction
```

Les paramètres des fonctions

Les fonctions permettent de faire un traitement sur des données.

Par exemple:

```
Début
        TraiterLesNombres(variable1, variable2)
Fin

Fonction TraiterLesNombres(variable1, variable2)
        Instructions à effectuer sur variable1 et variable2
Fin fonction
```

Les retours de fonctions

Les fonctions exécutent des traitements mais elles peuvent également retourner UNE et UNE SEULE variable.

```
Fonction TraiterLesNombres(variable1, variable2)
        somme = variable1 + variable2
        retourner somme
Fin fonction
```

La fonction va retourner la variable somme

Exemple 1 de pseudo-code

Set total to zero

Set grade counter to one

    While grade counter is less than or equal to ten

        Input the next grade

        Add the grade into the total

    End While

Set the class average to the total divided by ten

Print the class average.

Exemple 2 de pseudo-code

Exemple:

Si j’ai de l’essence dans la voiture

Tant qu’il reste de l’essence

J’avance à 2 KM à l’heure

Fin tant que

Fin si

Exemple 3 de pseudo-code

```
Variable X = 1
Tant que X < 100
        Afficher X
        X = X + 1
Fin tant que
```

Exemple 4 de pseudo-code

```
nomChien-1 = Cookie
ageChien-1 = 5
nomChien-2 = Alfred
ageChien-2 = 5
nomChien-3 = Boris
ageChien-3 = 3

Fonction afficherNomChien
        Si ageChien-1 > 3 Alors
                Afficher nomChien-1
        Fin Si
        Si ageChien-2 > 3 Alors
                Afficher nomChien-2
        Fin Si
        Si ageChien-3 > 3 Alors
                Afficher nomChien-3
        Fin Si
Fin fonction
```

Exemple 5 de pseudo-code

```
Variable nombre = 0
Tant que nombre != 100
        nombre = nombre + 1
        Afficher nombre
Fin tant que
```

Mot de la fin

Les algorithmes représentent la base de la programmation. Pour pouvoir effectuer de la programmation il faut comprendre et maîtriser les algorithmes!

Assurez-vous de maîtriser les éléments permettant de gérer le *flow* de traitement dans le pseudo-code

Les éléments du pseudo code			
Et	Ou	Fin	Débuter
Si	Alors	Tant que	Sinon
Fin si	Fin tant que	Fonction	Pour

Questions ?