Algorithmique I

GRETA - BTS SIO - 1ère année

Plan de Cours

Introduction

- Introduction à la Programmation
- Notion d'Algorithme, de Complexité et de Langage

Algorithmique I

- Algorithmes
- Types de Données, Variables et Constantes
- Opérateurs, Expressions et Affectations
- Structures de Contrôle

Algorithmique II

- Tableaux
- Procédures, Fonctions et Paramètres
- Variables Locales et Globales
- Structures
- Fichiers
- Récursivité

Algorithme

- Un algorithme est la description d'un processus de résolution d'un problème défini, rédigé dans un langage formalisé et qui produit un résultat en un temps fini
- C'est une succession d'instructions élémentaires qui, appliquée sur un ensemble de ressources (entrée), fournie la solution (sortie) au problème en question

Formalisme

- Un algorithme possède :
  - Un en-tête qui spécifie le nom de l'algorithme et son rôle. Il contient aussi les déclarations des ressources nécessaires au fonctionnement de l'algorithme
  - Un corps qui débute par "Début" et se termine par "Fin". Il contient les instructions indentées de l'algorithme.

Formalisme (suite)

- Exemple :

Algorithme mon_algo
[Rôle : …]
[*Ressources* : …]

Début

…

Fin

Type de Données

- Un type de données est une nature d'information
- Il caractérise un ensemble de valeurs possibles
- Il induit un ensemble d'opérations réalisables sur les informations de cette nature

Principaux Types de Données

- Les Entiers (entiers naturels signés)
- Les Réels (réels signés)
- Les Booléens (VRAI ou FAUX)
- Les Caractères (imprimables ou pas – ASCII) et les Chaînes de caractères
  (Les valeurs de ces 2 types de données sont délimités par le caractère " [double quote ou guillemet])

Variable

- Une variable est une entité nommée qui possède un type et une valeur
- Le type d'une variable caractérise l'ensemble des valeurs que peut prendre la variable et les opérations réalisables sur cette variable
- L'utilisation d'une variable dans un algorithme doit être précédée de sa déclaration

Variable (suite)

- La valeur d'une variable peut changer au cours du déroulement de l'algorithme
- Le type d'une variable est figé lors de sa déclaration et ne peut donc pas changer au cours du déroulement de l'algorithme

Déclaration d'une Variable

- Dans un algorithme, vous déclarez les variables dans l'en-tête de l'algorithme, précédées du mot clé " variable(s) "
- La syntaxe est *nom_variable : type*
- Exemples :

```
Variables
        a : Entier
        b, c : Réels
```

Constante

- Une constante est une variable dont la valeur reste figée durant le déroulement d'un algorithme
- Le type de la constante découle de sa valeur
- Sa déclaration sera vue ultérieurement

Règles de nommage

- Le nom d'un algorithme, d'une variable ou d'une constante doit respecter les règles suivantes :
  - Commencer par une lettre
  - Ne pas comporter de caractères spéciaux ou de ponctuation (caractère "espace", par exemple)
  - Ne pas être un mot du langage algorithmique (comme "algorithme", "début", "fin", "variable", "NON", "OU" etc.)

Opérateurs et Opérandes

- Un opérateur est un symbole d'opération qui agit sur une variable, ou qui permet d'effectuer des " calculs ".
  +, x, ÷, NON, OU... sont des opérateurs
- Une opérande est une entité utilisée par un opérateur. Dans " 5 - 3 ", " - " est l'opérateur; " 5 " et " 3 " sont les opérandes

Opérateurs

- Il existe plusieurs types d'opérateurs :
  - Les opérateurs arithmétiques
  - Les opérateurs de comparaison
  - Les opérateurs logiques
  - L'opérateur de concaténation
  - L'opérateur d'affectation ($\leftarrow$)

Opérateurs Arithmétiques

- Ils permettent d'effectuer des opérations arithmétiques entre opérandes numériques :
  - Opérations élémentaires : " + ", " - ", " × ", " / "
  - Changement de signe : " - "
  - Élévation à la puissance : " ^ "
  - Modulo : " Modulo " (ou " mod ")

Opérateurs de Comparaisons

- Ils permettent de comparer 2 opérandes et produisent une valeur booléenne
- Ils s'appuient sur des relations d'ordre :
  - Ordre naturel (entiers ou réels)
  - Ordre lexicographique ASCII (chaînes de caractères)
- Égalité (=), inégalité stricte (<, >), inégalité large (≤, ≥), différence (≠)

Opérateurs Logiques

- Les opérateurs logiques combinent des opérandes booléennes pour former des expressions logiques plus complexes :
  - Opérateur unaire NON
  - Opérateurs binaires ET, OU et OU_EXCLUSIF

Opérateur de Concaténation

- L'opérateur de concaténation permet " d'additionner " 2 chaînes de caractères
- Le résultat de cette " addition " est une chaîne de caractères constituée de la mise " bout à bout " des 2 chaînes concaténées
- Il peut être considéré comme un l'opérateur arithmétique d'addition pour les chaînes de caractères

Opérateur d'Affectation

- L'opérateur d'affectation est représenté par le symbole $\leftarrow$
- C'est l'opérateur qui confère une valeur à une variable, ou a une constante
- Exemple : " a $\leftarrow$ 3 " confère la valeur 3 à la variable " a ", ou à la constante " a "

Retour sur la Déclaration d'une Constante

- Dans un algorithme, vous déclarez les constantes dans l'en-tête de l'algorithme, précédées du mot clé " constante(s) "
- La syntaxe est *nom_constante* ← *valeur*
- Exemples :

```
constantes
        un_euro ← 6.55957
        pi  ← 3.1415927
```

Expression

- Une expression est une combinaison d'opérateur(s) et d'opérande(s)
- Durant le déroulement de l'algorithme, toutes les expressions sont évaluées
- L'évaluation de toute expression produit une valeur et donc, toute expression possède un type

Expression (suite)

- Exemples : Expression " A + B "
  - " A " et " B " sont les opérandes, " + " est l'opérateur
  - Si " A " et " B " sont des entiers, l'évaluation de " A+B " produira un entier
  - Si " A " vaut 3 et " B " vaut 5, l'évaluation de l'expression " A + B " produira la valeur 8

Priorité des Opérateurs

- A chaque opérateur est associé une priorité
- Lors de l'évaluation d'une expression, la priorité de chaque opérateur permet de définir l'ordre dans lequel les différentes opérations seront effectuées
- Pour lever l'ambiguïté ou modifier cet ordre, on utilise des parenthèses

Priorité des Opérateurs (suite)

- Ordre de priorité décroissante des opérateurs arithmétiques :
  - " ^ " (élévation à la puissance)
  - " - " (changement de signe)
  - " x " et " ÷ "
  - " Modulo "
  - " + " et " - "
  - " + " (concaténation)

Priorité des Opérateurs (suite)

- Ordre de priorité décroissante des opérateurs logiques :
  - " Non "
  - " Et "
  - " Ou "
  - " Ou exclusif "
- La question de l'ordre de priorité des opérateurs de comparaison ne se pose pas

Entrées / Sorties

- Elles réalisent les interactions entre l'utilisateur et le programme
- Entrée : saisie d'une information sur le clavier, auprès de l'utilisateur
- Sortie : Affichage d'une information sur l'écran, à destination de l'utilisateur

Entrées / Sorties (suite)

- Entrée : la syntaxe est variable Saisir(*variable*) la valeur saisie auprès de l'utilisateur étant affectée à *variable*
- Sortie : la syntaxe est Afficher(*message*, *expression*), le *message* est écrit, suivit de la valeur résultante de l'évaluation de l'*expression*

Entrées / Sorties (suite)

- Exemple :

```
Algorithme Double
Rôle : Calcul le double d'un entier saisi auprès de l'utilisateur
Variables a, b : entiers
Début
    Afficher("Entrez la valeur de a : ")
    Saisir(a)
    b ← a x 2
    Afficher("Le double de a vaut ", b)
Fin
```

Structures de Contrôle

- Une structure de contrôle permet de modifier le déroulement séquentiel d'une suite d'instructions
- Il y a deux familles de structures de contrôle :
  - les structures de contrôle alternatives
  - Les structures de contrôle itératives

Structures de Contrôle Alternatives

- Une structure de contrôle alternative permet d'introduire un choix, quant aux instructions exécutées dans un algorithme
- Ce choix s'effectue par l'évaluation d'une expression, dont le résultat conditionne l'exécution d'instructions spécifiques

Structures de Contrôle Alternatives (suite)

- Deux structures de contrôle alternatives :
  - Si ... Alors ... Sinon ...
  - Cas où ... Vaut : ...

L'instruction Si … Alors … Sinon …

- L'instruction Si … Alors … Sinon … conditionne l'interprétation des instructions d'un algorithme, au résultat d'une expression booléenne
- La syntaxe est :

Si *Expression booléenne* Alors
    Suite d'instructions (si Expression est vraie)
[Sinon
    Suite d'instructions (si Expression est fausse)]
Fin Si

L'instruction Si ... Alors ... Sinon ... (suite)

- Exemples :

```
Si A < B Alors
        Afficher("A < B")
Sinon
        Afficher("A ≥ B")
Fin si
```

```
Variable A : Entier

Si A modulo 2 = 1 Alors
        A ← A – 1
Fin Si
A ← A / 2
```

L'instruction Cas où … Vaut : …

- L'instruction Cas où … Vaut : … conditionne l'interprétation des instructions d'un algorithme, au résultat d'une expression
- Les différentes valeurs prises par l'expression sont décomposées en autant de cas disjoints que nécessaire
- Un cas " Autre ", facultatif, peut être ajouté à la structure

L'instruction Cas où … Vaut : … (suite)

- La syntaxe est :

Cas où *Expression* Vaut :
Val1 : Suite d'instructions
$Val2_1$, $Val2_2$, …, $Val2_n$: Suite d'instructions
…
Val_n : Suite d'instructions
[Autre : Suite d'instructions]
Fin Cas

Où " Val_i " représente une valeur constante et " Autre ", le cas exécuté si aucune des valeurs Val_i indiquées ne correspond au résultat de *Expression*

L'instruction Cas où … Vaut : … (suite)

- Exemples :

```
Cas où Lettre Vaut :
  "A", "E", "I", "O", "U", "Y" :
      Afficher("voyelle")
  Autre :
      Afficher("consonne")
Fin Cas
```

```
Cas où Mois Vaut :
  1, 3, 5, 7, 8, 10, 12 :
      Afficher("31 jours")
  4, 6, 9, 11 :
      Afficher("30 jours")
  2 :
      Afficher("28 ou 29 jours")
Fin Cas
```

Structures de Contrôle Itératives

- Une structure de contrôle itérative permet de répéter certaines instructions d'un algorithme, sans la nécessité de dupliquer ces instructions
- Cette répétition peut être définie :
  - D'une manière déterministe : le nombre d'exécution des instructions est indiqué en début de structure
  - D'une manière non déterministe : le nombre d'exécution des instructions est conditionnée par l'évaluation d'une expression booléenne

L'instruction Pour …

- Pour … est une structure de contrôle itérative déterministe. Sa syntaxe est :

  Pour *variable* ← *valeur_départ* à *valeur_arrivée*
  Instructions à répéter
  Fin pour

- Elle permet de répéter les instructions:
  - (*valeur_arrivée* - *valeur_départ* + 1) fois, si *valeur_arrivée* ≥ *valeur_départ*
  - zéro fois, si *valeur_arrivée* < *valeur_départ*

L'instruction Pour ... (suite)

- Par défaut, la valeur de la *variable* utilisée augmente de 1 unité à chaque passage sur l'instruction "Fin Pour". On dit que le pas d'incrémentation de la *variable* est 1
- Pour incrémenter *variable* d'une autre valeur, il faut indiquer cette valeur particulière de pas
- La syntaxe est alors :

  Pour variable ← *départ* à *arrivée* par pas de *valeur*

  ...

  Fin pour Remarque variable ← variable + *valeur*

L'instruction Pour ... (suite)

- Exemples :

```
Pour i ← 1 à 10
        Afficher(i)
Fin pour

Pour i ← 1 à 10 par pas de 2
        Afficher(i)
Fin pour
```

L'instruction Tant que …

- Tant que … est une structure de contrôle itérative non déterministe. Sa syntaxe est :

  Tant que *expression_booléenne* Faire
  Instructions à répéter
  Fin tant que

- Les instructions sont répétées, tant que l'évaluation de l'expression booléenne a pour valeur VRAI. Dès que cette valeur devient FAUX, les instructions cessent d'être répétées

L'instruction Tant que ... (suite)

- En conséquence, les instructions sont répétées jusqu'à ce que e*xpression_booléenne* devienne fausse
- Ce type de structure de contrôle est dit " a condition en tête ", car l'expression booléenne est évaluée avant l'exécution des instructions contenues dans la structure

L'instruction Tant que ... (suite)

- Exemples :

```
i ← 1
Tant que i < 11 Faire
        Afficher(i)
        i ← i + 1
Fin tant que
```

```
i ← 1
Tant que i < 11 Faire
        Afficher(i)
        i ← i + 2
Fin tant que
```

```
Variable n : Entier

Afficher("Entrer la valeur de n")
Saisir(n)

Tant que n ≤ 0
   Afficher(" Ré-entrer la valeur de n")
   Saisir(n)
Fin tant que
```

L'instruction Répéter … Jusqu'à ce que …

- Répéter … Jusqu'à ce que … est une structure de contrôle itérative non déterministe. Sa syntaxe est :

  Répéter

  Instructions à répéter

  Jusqu'à ce que *expression_booléenne*

- Les instructions sont répétées, jusqu'à ce que l'évaluation de l'expression booléenne ait pour valeur VRAI.

L'instruction Répéter … Jusqu'à ce que … (suite)

- En conséquence, les instructions sont répétées tant que *expression_booléenne* est fausse
- Ce type de structure de contrôle est dit " a condition en queue ", car l'expression booléenne est évaluée après l'exécution des instructions contenues dans la structure

L'instruction Répéter ... Jusqu'à ce que ... (suite)

- Exemple : Algorithme d'Euclide

```
variable a, b : Entiers
variable reste : Entier

Saisir(a)
Saisir(b)
Répéter
        reste ← a modulo b
        a ← b
        b ← reste
Jusqu'à ce que reste = 0
Afficher("le pgcd vaut ", a)
```

Remarques

- Une structure Tant que … voit les instructions qu'elle contient exécutées 0 ou n fois
- Une structure Répéter … Jusqu'à ce que … voit les instructions qu'elle contient exécutées 1 ou n fois