Algorithmique II

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

Tableaux

- Un tableau est une structure particulière de variable : cette structure permet à une seule variable de stocker plusieurs valeurs
- Chaque valeur stockée est repérée par un, ou plusieurs, indices
- Cet (ou ces) indice(s) repère(nt) la position d'une valeur particulière, dans l'ensemble des valeurs stockées par le tableau

Tableaux à une dimension

- Dans un tableau à une dimension, chaque valeur stockée est repérée par un unique indice
- L'indice d'une valeur est un entier définissant la position de cette valeur, dans l'ensemble des valeurs stockées

Tableaux à une dimension (suite)

- La syntaxe de déclaration d'un tableau à une dimension est la suivante :

  nom_tableau [*nbre_valeurs*] : Tableau de *type*

  - Où *nbre_valeurs* est le nombre de valeurs pouvant être stockées dans ce tableau
  - Et *type*, le type de chacune de ces valeurs. Car toutes les valeurs d'un tableau sont de même type

Tableaux à une dimension (suite)

- En algorithmique, la première valeur d'un tableau à une dimension est à l'indice 1. Si ce tableau stocke n valeurs, la dernière valeur est donc à l'indice n
- Pour accéder a l'une des valeurs d'un tableau à une dimension, la syntaxe est :

nom_tableau [*indice_valeur*]

Tableaux à une dimension (suite)

- Exemples :

Variable T[10] : Tableau d'entiers

```
T[1] ← 2
A ← 3
T[A] ← 45
T[i] ← T[i + 1]
T[2 * i – 1] ← …
T[T[j]] ← …
```

```
Afficher(T[1])
Afficher(T[A])
Pour i ← 1 à 10
    T[i] ← i
    Afficher(T[i])
Fin Pour
```

Tableaux multidimensionnels

- Un tableau peut être multidimensionnel : il est alors nécessaire d'utiliser plusieurs indices pour repérer une valeur dans ce tableau
- La syntaxe de déclaration d'un tel tableau est la suivante :

nom_tableau [*nb*$_1$][*nb*$_2$][…][*nb*$_n$] : Tableau de *type*

Où nb_1, nb_2, …, nb_n représentent le nombre de valeurs stockées dans chaque dimension du tableau

Tableaux multidimensionnels (suite)

- Pour accéder a l'une des valeurs d'un tableau multidimensionnel, la syntaxe est :

  $nom\_tableau\ [indice_1][indice_2]...[indice_n]$

- Exemple :

```
Variable T[10][10] : Tableau de réels

Pour i ← 1 à 10
    Pour j ← 1 à 10
        T[ i ][ j ] ← 0.0
    Fin pour
Fin pour
```

Procédures et Fonctions

- Une procédure, ou une fonction, est un groupe d'instructions nommé, indépendant du reste de l'algorithme mais utilisé par lui
- Pour exécuter les instructions contenues dans une procédure, ou une fonction, l'algorithme effectue un appel à cette procédure, ou à cette fonction

Procédures et Fonctions (suite)

- Les instructions qui suivent l'appel à une procédure, ou à une fonction, ne sont exécutées que lorsque les instructions de la procédure, ou de la fonction, appelée ont été exécutées
- Même en cas d'appel à une procédure, ou à une fonction, le déroulement d'un algorithme reste donc linéaire

Procédures et Fonctions (suite)

- Les procédures et les fonctions participent du principe méthodologique : abstraire, décomposer, combiner et itérer
- Elles permettent de décomposer un problème en sous-problèmes disjoints et indépendants
- Elles facilitent la réutilisation des instructions d'un algorithme, pour la résolution d'un autre problème

Structure d'un Algorithme

- Algorithme …

  Constantes : …
  Variables : …

  [procédures et fonctions]

  Début
  …
  Fin

Le nom de l'algorithme

Les constantes et les variables

Les procédures et les fonctions de l'algorithme

L'algorithme " principal " contenant des appels aux procédures et aux fonctions ci-dessus

Terminologie

- Le bloc d'instructions qui appelle une procédure, ou une fonction, est nommé " appelant "
- Le bloc procédure, ou fonction, en question est nommée " appelé "
- Dans la majorité des langages de programmation, l'appelé doit être au dessus de l'appelant

Procédures

- Une procédure est un bloc d'instructions nommé. Sa syntaxe algorithmique est :

```
Procédure nom_procédure (…)
…
Début
        …
Fin Procédure
```

Fonctions

- Une fonction est une procédure qui renvoie une unique valeur à l'appelant. Sa syntaxe algorithmique est :

```
Fonction nom_fonction (…) : type de la valeur retournée
…
Début
        …
        Retourner(valeur_retournée)
Fin Fonction
```

Fonctions (suite)

- L'instruction " Retourner (*valeur)* " retourne, à l'appelant, la valeur prise par la fonction
- Cette instruction met fin à l'exécution de la fonction
- Toute fonction doit posséder au moins une instruction " Retourner ", mais elle peut en posséder plusieurs

Appel

- L’appel à une procédure s’effectue par le nom de cette procédure
- L'appel à une fonction s'effectue par le nom de cette fonction, suivi de parenthèses obligatoires. La valeur retournée par la fonction doit être utilisée

Variables Locales et Globales

- Une procédure ou une fonction peut posséder ses propres constantes et ses propres variables
- Celles-ci sont inconnues du reste de l'algorithme (donc inutilisables par le reste de l'algorithme)
- Elles sont dites " locales ", à la procédure ou à la fonction

Variables Locales et Globales (suite)

- Les constantes et variables définies au niveau de l'algorithme principal sont connues de tout l'algorithme
- Elles sont donc aussi connues de chaque procédure ou fonction (donc utilisables par ces procédures ou ces fonctions)
- Elles sont dites " globales ", à l'algorithme

Portée

- La portée d'une variable (constante) est, dans un algorithme, l'endroit où la variable (constante) est connue et donc, utilisable
- La portée d'une variable (constante) locale est le bloc d'instructions où elle est déclarée
- La portée d'une variable (constante) globale est l'algorithme tout entier

Portée (suite)

- Les variables (constantes) locales peuvent avoir le même nom que des variables (constantes) globales
- Dans ce cas, les variables (constantes) locales " masquent " les variables (constantes) globales, dans les portées communes. On parle alors de phénomène " d'éclipse "

Portée (suite)

- Exemple :

```
Algorithme …
Variable i, j : Entier
        …
        Procédure P()
        Variable i : Réel
        Début
                i ← 1.0
                j ← 2
        Fin Procédure
        …
Début
        i ← 0
        j ← 1
        …
Fin Algorithme
```

Durée de Vie

- La durée de vie d'une variable (constante) est la période durant laquelle la variable (constante) existe et donc, la période durant laquelle la valeur associée à cette variable (constante) existe
- La durée de vie est liée à la portée

Durée de Vie (suite)

- La durée de vie d'une variable (constante) globale est la durée nécessaire à l'exécution de l'algorithme tout entier
- La durée de vie d'une variable (constante) locale à un bloc est la durée nécessaire à l'exécution du bloc : la vie d'une variable locale prend donc fin avec l'exécution du bloc d'instructions où elle est déclarée

Procédures avec données locales

- Avec des données locales, la syntaxe algorithmique d'une procédure est :

Procédure *nom_procédure* (…)
constantes : …
variables : …
Début
…
Fin Procédure

Fonctions avec données locales

- Avec des données locales, la syntaxe algorithmique d'une fonction est :

Fonction *nom_fonction* (…) : type de la valeur retournée
constantes : …
variables : …
Début
    …
    Retourner (*valeur_retournée)*
Fin Fonction

Paramètres

- Un paramètre de procédure, ou de fonction, est l'information nécessaire à l'exécution de cette procédure, ou de cette fonction
- Il permet de " paramétrer " l'exécution de la procédure, ou de la fonction, en la rendant plus généraliste et donc, plus " puissante "

Paramètres (suite)

- Un paramètre de procédure, ou de fonction, est nommé et typé
- Il est indiqué, entre parenthèses, derrière le nom de la procédure ou de la fonction
- S'il y a plus d'un paramètre, ceux-ci sont séparés par une virgule

Paramètres (suite)

- Exemples :

```
Procédure A(i : Entier)
…
Début
        …
Fin

Fonction B(j : Réel, k : Booléen) : Entier
…
Début
        …
Fin
```

Passages de Paramètres

- Le passage de paramètre est la transmission d'un paramètre à une procédure, ou à une fonction, lors de l'appel de celle-ci
- Il y a trois types de passage de paramètres :
  - Par valeur : On parle alors de paramètre "donnée"
  - Par référence : On parle alors de paramètre "résultat"
  - Un mélange des 2 précédents : Le paramètre est à la fois un paramètre donnée et résultat

Paramètres "données"

- Les paramètres "données" transmettent une valeur à une procédure, ou à une fonction
- La valeur reçue par la procédure, ou la fonction, est une copie de la valeur ayant servie à l'appel
- Toute modification effectuée sur la valeur de ce paramètre par la procédure ou la fonction, n'a aucune incidence sur la valeur originale

Paramètres "données" (suite)

- Par défaut, tous les paramètres de procédure ou de fonction sont des paramètres "données"
- Mais on peut volontairement indiquer qu'un paramètre est un paramètre donnée, en faisant précéder son nom par la lettre "D"

Exemples : Procédure A(i : Entier)
est identique à Procédure A(D i : Entier)

Paramètres "résultats"

- Les paramètres "résultats" transmettent une référence à une procédure, ou à une fonction
- La procédure, ou la fonction, " reçoit " la variable ayant servie à l'appel
- Toute modification de la valeur de ce paramètre par la procédure ou la fonction,
  est donc une modification de la valeur de la variable ayant servie à l'appel

Paramètres résultats (suite)

- Par défaut, tous les paramètres de procédure ou de fonction étant des paramètres données, il faut spécifiquement indiquer qu'un paramètre est un paramètre "résultat" en faisant précéder son nom de la lettre "R"

Exemples : Procédure A1(R i : Entier)
Procédure A2(i : Entier, R j : Entier)
est identique à Procédure A2(D i : Entier, R j : Entier)

Retour sur l'Appel

- L'appel à une procédure s'effectue par le nom de la procédure et, s'ils existent, les paramètres d'appel entre parenthèses, séparés par des virgules

- Exemples d'appels :

```
procédure_un
procédure_deux(1)
procédure_trois(1, A)
procédure_quatre(A, B, 3.87)
```

Retour sur l'Appel (suite)

- L'appel à une fonction s'effectue par le nom de la fonction suivit de parenthèses et, s'ils existent, les paramètres d'appel placés entre ces parenthèses, séparés par des virgules

- Exemples d'appels :

  … ← fonction_un()
  Si fonction_deux(1) …
  Tant que fonction_trois(1, A) …
  Au cas où fonction_quatre(A, B, 3.87) vaut :

Retour sur l'Appel (suite)

- Lors d'un appel à une procédure, ou à une fonction, avec paramètres :
  - Un paramètre "donnée" est une valeur (constante, valeur de variable, valeur d'une expression ou celle retournée par une fonction)
  - Un paramètre "résultat" ne peut être qu'une variable

Structures

- Les structures (ou enregistrements) sont des types de variables nommés, structurés en plusieurs éléments (ou champs)
- Chaque élément d'une structure est défini par son nom et son type
- Une variable, qui a pour type une structure, est dite " structurée "

Structures (suite)

- Exemple :

```
Structure Personne
        Prénom : Chaîne de caractères
        Nom : Chaîne de caractères
        Age : Entier
Fin Structure

Variable Moi, Toi : Personne
```

Structures (suite)

- Pour accéder aux éléments d'une variable structurée, on utilise la notation " pointée "
- Exemples :

```
Moi.Nom                 = "Tarzan"
Toi.Nom                 = "Jane"

Doyenne.Nom  = "Rodriguez"
Doyenne.Prénom          = "Juana"
Doyenne.Age             = 125
```

Structures (suite)

- Les variables structurées permettent de réduire, et de structurer, l'utilisation des variables dans un algorithme
- Elles s'utilisent comme des variables " simples " (passage de paramètre à une procédure ou à une fonction, valeur retournée par une fonction, comme type de tableau…)

Fichiers

- Les variables d'un programme sont allouées en mémoire centrale. Elles sont donc "volatiles", car elles disparaissent à la terminaison du programme qui les a créées, ou à l'arrêt de la machine qui abrite ce programme
- Les fichiers sont des dispositifs de stockage de l'information sur supports "permanents" : ils perdurent au-delà de l'arrêt du programme qui les a créés ou de la machine

Familles de Fichiers

- Il y a deux familles de fichiers : les fichiers textes et les fichiers binaires
- Dans un fichier texte, l'information est stockée sous sa forme textuelle
  Exemple : la valeur réelle 0,12345678 sera stockée sous la forme "`0,12345678`" ou "`0.12345678`"
- Dans un fichier binaire, l'information est stockée sous sa forme binaire
  Exemple : la valeur réelle 0,12345678 sera stockée sous sa forme binaire, soit `3DFC D6E9` en codage IEEE754

Type d'Accès aux fichiers

- Il existe deux types d'accès à un fichier : l'accès séquentiel et l'accès direct (ou aléatoire)
- L'accès séquentiel est plutôt réservé aux fichiers textes
- L'accès direct est plutôt réservé aux fichiers binaires

Fichiers à Accès Séquentiel

- Les enregistrements d'un fichier à accès séquentiel sont lus les uns à la suite des autres : pour atteindre un enregistrement particulier, il faut lire les enregistrements qui le précèdent
- Ce type d'accès est nécessaire lorsque la taille de chaque enregistrement du fichier est différente, ou inconnue
- Par exemple, un fichier texte

Fichiers à Accès Direct

- Les enregistrements d'un fichier à accès direct sont lus directement : on peut atteindre un enregistrement particulier sans avoir lu aucun autre enregistrement au préalable
- Ce type d'accès est possible lorsque la taille de chaque enregistrement est identique et/ou connue
- Par exemple, un fichier dont chaque enregistrement est une structure de taille fixe

Fichiers (suite)

- Pour travailler dans un fichier, quelque soit l'activité, il faut:
  - L'ouvrir, en indiquant ce que l'on va y faire
  - Faire ce que l'on a à y faire
  - Le refermer
- Dans un fichier, les deux principales activités sont la lecture et l'écriture d'enregistrements
  Remarques : créer un enregistrement revient à écrire cet enregistrement; La suppression d'un enregistrement n'existe pas…

Fichiers (suite)

- Dans le cas d'une lecture séquentielle, savoir que la fin du fichier est atteinte est indispensable
- Dans le cas d'un accès direct, se positionner sur un enregistrement précis est indispensable

Récursivité

- La récursivité est le fait de décrire un processus dépendant de données, en faisant appel à ce même processus sur d'autres données plus "simples"
- En informatique, un algorithme qui contient un appel à lui-même est dit récursif
- Par extension, une fonction (ou une procédure) qui contient un appel à elle-même est dite récursive

Récursivité (suite)

- La récursivité peut être directe si un algorithme, ou une fonction, s'appelle lui-même
- La récursivité peut-être indirecte (ou croisée) si un algorithme (ou une fonction) en appelle un second, qui appelle le premier
- Dans tout les cas, il existe une clause de finitude permettant de mettre fin aux appels successifs

Récursivité (suite)

- Exemple : calcul récursif d'une factorielle

  Pour x, entier positif ou nul :
  x! = 1, si x = 0;
  x! = x*(x-1)*(x-2)*(x-3)*…*3*2*1
  C'est une définition itérative de la factorielle

  Que l'on peut transformer en :
  x! = 1, si x = 0;
  x! = x*(x-1)!
  Et c'est une définition récursive de la factorielle

Récursivité (suite)

- Attention !

  Si un algorithme récursif permet de résoudre un problème complexe plus "simplement" qu'un algorithme itératif, son implémentation dans un langage de programmation peut s'avérer catastrophique en terme de temps d'exécution, ou d'occupation mémoire

  Exemple : calcul de la suite de Fibonacci