مدرسة علوم المعلومات
ⵜⵉⵎⵔⵔⴰⵜ ⵏ ⵜⵓⵙⵙⵏⴰ ⵏ ⵜⵣⵔⴼⵜ
ECOLE DES SCIENCES DE L'INFORMATION
www.esi.ac.ma

Algorithmique

6. Procédures et Fonctions

Sommaire

Rappel

Rappel

- Algorithme informatique
  - Suite d'instructions ordonnée qui décrit de façon exhaustive les différentes étapes à suivre par un processeur pour résoudre un problème donné en un temps fini

Rappel

- Pseudo-code algorithmique

ALGORITHME *nom_de_l'algorithme*

<partie des déclarations>

DEBUT

<partie des instructions>

//commentaire

FIN

Rappel

- Un programme n'est pas purement séquentiel ➔ nécessité d'avoir des **structures de contrôle**

1. Les structures alternatives (tests)
2. Les structures itératives (boucles)

Rappel

- Tableau statique
  - Séquence de données de **même type**, chacune référencée par un nombre appelé **indice** (ou index)
  - Désigné par
    - Son nom
    - Le type de ses éléments
    - Sa taille (i.e. le nombre de ses éléments)

Rappel

- Tableau dynamique
  - Séquence de données de **même type**, chacune référencée par un nombre appelé **indice** (ou index), dont la **taille est variable** et peut changer lors de l'exécution
  - L'opération de redimensionnement permet de fixer la taille du tableau dynamique. Elle est effectuée par l'instruction ***Redim***

Rappel

- Tableau à 2 dimensions
  - Séquence de données de **même type**, chacune référencée par **deux indices**
  - Désigné par
    - Son nom
    - Le type de ses éléments
    - Sa taille n x m

Problématique

- Généralement, un algorithme comporte plusieurs parties, chacune effectuant un traitement donné
  - Exemple : gestion des étudiants, gestion des clients, gestion de stock…
  - ➔ Programme trop long et/ou trop complexe

Problématique

- Plusieurs concepteurs sont amenés à collaborer sur un même algorithme, chacun sur une ou plusieurs parties
- Nécessité de réutiliser le code
  - Exemple : algorithme de recherche dichotomique, algorithme de tri des tableaux …

Problématique

➔ L'idéal est de décomposer un algorithme en plusieurs « petits » algorithmes

- Chacun traite une partie du problème
- Au besoin, ils s'appellent entre eux
- ➔ Utilisation de sous-procédures et de fonctions

Intérêt et motivation

Intérêt et motivation

- Pourquoi découper un seul algorithme en plusieurs « petits » algorithmes ?
  - Améliorer la lisibilité de l'algorithme et, par conséquent celle du programme qui sera implémenté
  - Faciliter la maintenance de l'algorithme / programme
  - Favoriser la réutilisation de l'algorithme / programme

```
ALGORITHME exemple
VAR tab1[10], tab2[10], tab[10] : réel
DEBUT
 //remplir le tableau tab1
 POUR i ← 0 à 9
    Lire(tab1[i])
 i SUIVANT
 //remplir le tableau tab2
 POUR i ← 0 à 9
    Lire(tab2[i])
 i SUIVANT
 //remplir le tableau tab3
 POUR i ← 0 à 9
    Lire(tab3[i])
 i SUIVANT
  //trier le tableau tab1
  POUR i ← 0 à 9
    POUR j ← i + 1 à 10
       SI tab1[j] < tab1[i] ALORS
            x ← tab1[i]
            tab1[i] ← tab1[j]
            tab1[j] ← x
       FINSI
    j SUIVANT
  i SUIVANT
```

```
 //trier le tableau tab2
 POUR i ← 0 à 9
   POUR j ← i + 1 à 10
      SI tab2[j] < tab2[i] ALORS
           x ← tab2[i]
           tab2[i] ← tab2[j]
           tab2[j] ← x
      FINSI
   j SUIVANT
 i SUIVANT

 //trier le tableau tab3
 POUR i ← 0 à 9
   POUR j ← i + 1 à 10
      SI tab3[j] < tab3[i] ALORS
           x ← tab3[i]
           tab3[i] ← tab3[j]
           tab3[j] ← x
      FINSI
   j SUIVANT
 i SUIVANT

 //autres instructions du programme
 …

FIN
```

Intérêt et motivation

```
ALGORITHME exemple
VAR tab1[10], tab2[10], tab[10] : réel
…
DEBUT
  //remplir les tableaux
  Remplir(tab1,10)
  Remplir(tab2, 10)
  Remplir(tab3, 10)

  //trier les tableaux tab1
  Trier(tab1,10)
  Trier(tab2, 10)
  Trier(tab3, 10)
FIN
```

Intérêt et motivation

- Le corps de l'algorithme est appelé procédure principale
- Les instructions qu'on regroupe pour effectuer un traitement particulier sont appelées **sous-procédures** ou **fonctions**
  - On fait appel à ces sous-procédures ou fonctions à partir de la procédure principale

Sous-procédures

Définition

- Une **sous-procédure** est un ensemble d'instructions décrivant une action simple ou complexe
- Elle est définie par
  - Son nom
  - Ses arguments (ou paramètres)

Syntaxe

- Syntaxe de la définition

PROCÉDURE *nomProcédure(arg1:type, …, argN:type)*

VAR …

DÉBUT

…

FINPROCÉDURE

Syntaxe

- Syntaxe de l'appel

```
DEBUT
//instructions
…
nomProcédure(arg1, …, argN)
//autres instructions
…
FIN
```

Arguments

- Les arguments d'une sous-procédure sont mis entre parenthèses lors de sa déclaration et des appels
- Ils servent à transmettre des informations de la procédure principale (ou le programme appelant) à la sous-procédure

Arguments

- Une sous-procédure peut avoir 0 ou n arguments
  - Même sans arguments, on garde les parenthèses en les laissant vides
- Les arguments sont passés soit **par valeur** soit **par adresse**

Passage des arguments

- Passage par valeur
  - Mode de passage par défaut
  - Syntaxe de la définition

PROCÉDURE *nomProcédure(arg1:type, …, argN:type)*

VAR …

DÉBUT

…

FINPROCÉDURE

Passage des arguments

- Passage par valeur
  - Mode de passage par défaut
  - Syntaxe de la définition
  - Syntaxe de l'appel

```
DEBUT
//instructions
…
nomProcédure(var1, …, varN)
//autres instructions
…
FIN
```

Passage des arguments

- Passage par valeur
  - Lors de l'appel, la sous-procédure crée N nouvelles variables : arg1,…, argN
  - Chaque nouvelle variable va recevoir la **valeur** des variables var1,…, varN (dans l'ordre de déclaration) ➔ elle sera une copie du paramètre passé par valeur et sera détruite à la fin de l'exécution de la sous-procédure
    - **Les variables var1,…,varN ne sont pas modifiées**

Passage des arguments

- Passage par valeur

```
ALGORITHME exemple

VAR n : entier

PROCÉDURE Tripler(arg : entier)
DÉBUT
arg ← arg*3
FINPROCÉDURE

DEBUT
  n ← 2
  Tripler(n)
  Afficher(n)
FIN
```

Passage des arguments

- Passage par valeur
  - Avantage
    - Les variables du programme appelant sont protégées contre toute modification de leur contenu
  - Inconvénients
    - Les variables utilisées ne peuvent être que des paramètres en entrée, jamais en sortie
    - Utilisation de la mémoire

Passage des arguments

- Passage par adresse
  - Syntaxe de la définition

PROCÉDURE *nomProcédure(arg1:type*, …, argN:type*)*

VAR …

DÉBUT

…

FINPROCÉDURE

Passage des arguments

- Passage par adresse
  - Syntaxe de la définition
  - Syntaxe de l'appel

```
DEBUT
//instructions
…
nomProcédure(&var1, …, &varN)
//autres instructions
…
FIN
```

Passage des arguments

- Passage par adresse
  - Lors de l'appel, la sous-procédure crée N nouvelles variables : arg1,…, argN
  - Chaque nouvelle variable va recevoir l'**adresse** des variables var1,…, varN (dans l'ordre de déclaration) ➔ toute modification de arg1,…, argN sera répercutée sur arg1,…, argN
    - **Les variables var1,…,varN peuvent être modifiées**

Passage des arguments

- Passage par adresse

```
ALGORITHME triplerAlgo

VAR n : entier

PROCÉDURE Tripler(arg:entier*)
DÉBUT
arg ← arg*3
FINPROCÉDURE

DEBUT
  n ← 2
  Tripler(&n)
  Afficher(n)
FIN
```

Passage des arguments

- Passage par adresse
  - Avantage
    - Grâce aux pointeurs, les variables du programme peuvent être modifiées pour répondre à un besoin particulier ➔ les variables utilisées peuvent être des paramètres en entrée et en sortie
  - Inconvénient
    - Risque d'écraser par erreur les variables du programme appelant

Passage des arguments

- Remarques
  - Un tableau est toujours passé par adresse, et son identificateur représente son adresse

Passage des arguments

- Remarques

```
ALGORITHME nomAlgorithme

VAR notes[10] : réel

PROCÉDURE RemplirTableau(tab[ ]:réel, n: entier)
VAR i : entier
DÉBUT
  POUR i ← 0 à n
     Afficher("Donner la valeur de l'élément N° " , i+1)
     Lire(tab[i])
  i SUIVANT
FINPROCÉDURE

DEBUT
    RemplirTableau(notes, 10)
FIN
```

Passage des arguments

- Remarques
  - Un tableau est toujours passé par adresse, et son identificateur représente son adresse
  - Une sous-procédure peut faire appel à une autre sous-procédure
  - Une sous-procédure peut faire appel à elle-même ➔ **récursivité**

Exercices

- Écrire une sous-procédure qui permet d'afficher les nombres de 0 à 100

Exercices

```
ALGORITHME algoExercice

PROCÉDURE AfficherNombres()
VAR i : entier
DÉBUT
   POUR i ← 0 à 101
      Afficher(i)
   i SUIVANT
FINPROCÉDURE

DÉBUT
   AfficherNombres()
FIN
```

Exercices

- Écrire une sous-procédure qui calcule et affiche la somme et le produit de trois nombres réels passés en paramètre
  - Effectuer un appel depuis la procédure principale

Exercices

```
ALGORITHME algoExercice

VAR a, b, c: réel

PROCÉDURE SommeEtProduit(x, y, z : réel)
VAR somme, produit : réel
DÉBUT
      somme ← x + y + z
      produit ← x*y*z
      Afficher(somme, produit)
FINPROCÉDURE

DÉBUT
   Lire(a,b,c)
   SommeEtProduit(a,b,c)
FIN
```

Exercices

- Modifier la sous-procédure précédente pour stocker la somme et le produit dans des variables accessibles à partir de la procédure principale

Exercices

```
ALGORITHME algoExercice

VAR a, b, c, somme, produit: réel

PROCÉDURE SommeEtProduit(x, y, z:réel, s:réel*, p:réel*)
VAR somme, produit : réel
DÉBUT
   s ← x + y + z
   p ← x*y*z
   Afficher(somme, produit)
FINPROCÉDURE

DÉBUT
    Lire(a,b,c)
    SommeEtProduit(a,b,c, &somme, &produit)
FIN
```

Exercices

- Écrire une sous-procédure qui permet de permuter les valeurs de deux variables

Exercices

```
ALGORITHME algoExercice

VAR a, b: réel

PROCÉDURE Permutation(x: réel*, y:réel*)
VAR z : réel
DÉBUT
   z ← x
   x ← y
   y ← z
FINPROCÉDURE

DÉBUT
    Lire(a,b)
    Permutation(&a, &b)
FIN
```

Exercices

- Écrire une sous-procédure qui permet de remplacer un réel par sa valeur absolue

Exercices

```
ALGORITHME algoExercice

VAR a: réel

PROCÉDURE ValeurAbsolue(x: réel*)
DÉBUT
   SI x < 0 ALORS
      x ← −x
   FINSI
FINPROCÉDURE

DÉBUT
    Lire(a)
   ValeurAbsolue(&a)
FIN
```

Fonctions

Définition

- Une **fonction** est une sous-procédure particulière qui **retourne un résultat** à la procédure principale (ou le programme appelant)
- Elle est définie par
  - Son nom
  - Ses arguments (ou paramètres d'entrée)
  - Son type de retour

Syntaxe

- Syntaxe de la définition

FONCTION *nomFonction(arg1:type, …, argN:type)* : **type**

VAR …

VAR resultat : **type**

DÉBUT

…

Retourner resultat

FINFONCTION

Syntaxe

- Syntaxe de l'appel

```
DEBUT
//instructions
…
var ← nomFonction(arg1, …, argN)
//autres instructions
…
FIN
```

Arguments

- Une fonction sans arguments d'entrée s'écrit avec les parenthèses vides ()
- Les arguments sont utilisés de la même manière que pour les sous-procédures

Fonctions prédéfinies

- Les langages de programmation offrent des fonctions prédéfinies qui servent à effectuer des traitements qui ne peuvent être effectués autrement ou qui sont trop complexes à écrire par le programmeur
- Elles prennent comme argument des chaînes de caractères, des nombres, des tableaux…

Fonctions prédéfinies

- Exemples
  - **Length**(*texte:chaîne*) : retourne le nombre de caractères d'une chaîne de caractères
  - **Mid**(*texte:chaîne, car:caractère, n: entier*) : extrait une chaîne de caractère à partir de *texte*. Cette chaîne commence au caractère *car* et de faisant *n* caractères de long
  - **Asc**(*car:caractère*) : retourne le code ASCII d'un caractère
  - **Car**(*code:entier*) : retourne le caractère correspondant à un code ASCII

Fonctions prédéfinies

- Exemples
  - **Entier**(*x:réel*) : retourne la partie entière d'un nombre
  - **ConvNum**(*texte:chaîne*) : convertit un nombre qui est saisi et stocké comme une chaîne de caractères au type réel
  - **ConvChaine**(*x:réel*) : convertit un nombre qui est saisi et stocké comme nombre au type chaîne de caractères

Exercices

- Écrire une fonction qui calcule la somme de trois nombres réels passés en paramètre
  - Effectuer un appel depuis une procédure principale

Exercices

```
ALGORITHME algoExercice

VAR a, b, c, somme: réel

FONCTION Somme(x, y, z:Réel) : réel
VAR somme: réel
DÉBUT
   somme ← x + y + z
   Retourner somme
FINFONCTION

DÉBUT
   Lire(a,b,c)
   somme ← Somme(a,b,c)
FIN
```

Exercices

```
ALGORITHME algoExercice

VAR a, b, c, somme: réel

FONCTION Somme(x, y, z:réel) : réel
DÉBUT
   Retourner x + y + z
FINFONCTION

DÉBUT
   Lire(a,b,c)
   somme ← Somme(a,b,c)
FIN
```

Exercices

- Écrire une fonction qui permet de calculer la valeur absolue d'un réel
  - Effectuer un appel depuis une procédure principale

Exercices

```
ALGORITHME algoExercice

VAR a: réel

FONCTION ValeurAbsolue(x: réel) : réel
DÉBUT
   SI x < 0 ALORS
      x ← −x
   FINSI
   Retourner x
FINFONCTION

DÉBUT
    Lire(a)
    a ← ValeurAbsolue(a)
FIN
```

Exercices

- Quels sont les problèmes de cet algorithme ?

Exercices

```
ALGORITHME algoExercice
VAR a, b : entier
VAR c : caractère

FONCTION ValeurAbsolue(x: réel) : réel
DÉBUT
   SI x < 0 ALORS
      x ← −x
   FINSI
   Retourner x
FINFONCTION


DÉBUT
   Lire(a)
   Lire(c)
   b ← ValeurAbsolue(10*a − 20)
   a ← ValeurAbsolue(c)
   ValeurAbsolue(a) ← 0
FIN
```

Exercices

- Écrire une fonction qui permet de vérifier si un tableau est trié avec un ordre croissant ou non trié

```
FONCTION TableauTrie(tab[ ]: réel, taille: entier) : booléen
VAR i : entier
VAR trie : booleen
DÉBUT
  trie ← VRAI
  i ← 0
  TANTQUE trie ET i < taille FAIRE
    SI tab[i] > tab[i+1] ALORS
        trie ← FAUX
    FINSI
    i ← i + 1
  FINTANTQUE
  Retourner trie
FINFONCTION
```

Protée des variables

Définition

- La **portée d'une variable** est l'ensemble des sous-programmes (sous-procédures et fonctions) où elle est connue et peut donc être utilisée
- Une variable peut être
  - Locale
  - Globale

Variable locale

- Une variable **locale** est une variable qui est déclarée au sein d'un sous-programme et qui, par conséquent, ne peut être utilisée que dans ce sous-programme

Variable locale

- Une variable **locale** est une variable qui est déclarée au sein d'un sous-programme et qui, par conséquent, ne peut être utilisée que dans ce sous-programme

```
PROCÉDURE AfficherNombres()
VAR i : entier
DÉBUT
   POUR i ← 0 à 101
      Afficher(i)
   i SUIVANT
FINPROCÉDURE
```

Variable globale

- Une variable **globale** est une variable qui est déclarée au sein de la procédure principale
- Toute sous-procédure ou fonction appelée par la procédure principale connaît cette variable et peut l'utiliser

Variable globale

- Une sous-procédure ou fonction peut déclarer une variable globale qui, une fois créée, peut être utilisée par tous les autres sous-programmes de la procédure principale

Variable globale

- Une sous-procédure ou fonction peut déclarer une variable globale qui, une fois créée, peut être utilisée par tous les autres sous-programmes de la procédure principale

```
PROCÉDURE AfficherNombres()
VAR GLOBALE i : entier
DÉBUT
   POUR i ← 0 à 101
      Afficher(i)
   i SUIVANT
FINPROCÉDURE
```

Portée des variables

- Si un sous-programme déclare une variable locale qui a le même identificateur qu'une variable globale
  - La variable locale est utilisée par le sous-programme où elle est définie
  - La variable globale devient inaccessible par ce sous-programme

مدرسة علوم المعلومات
ⵜⵉⵎⵔⵔⴰⵜ ⵏ ⵜⵓⵙⵙⵏⴰ ⵏ ⵜⵣⵔⴼⵜ
ECOLE DES SCIENCES DE L'INFORMATION
www.esi.ac.ma

Algorithmique

6. Procédures et Fonctions