Introduction à l'algorithmique : Procédures et fonctions

Rappels, procédures, fonctions, arguments, portée, récursivité

Philippe Pasquier

Précisions

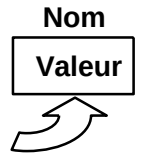
- **Pour les examens :**
  - **Écrits, en salle, sans machines (aucune)**
  - **Tous les documents sont autorisés**
  - **Types de connaissances :**
    - **Savoir synthétiser la matière**
    - **Savoir écrire un programme**
    - **Savoir le commenter**
    - **Savoir lire et comprendre un programme**
    - **Savoir rédiger la trace d'exécution d'un programme donné**
    - **Savoir présenter ses réponses avec clarté**

Précisions

- **Pour le travail pratique numéro 1 :**
  - **Le sujet sera sur le site Web du cours mercredi après-midi**
  - **Vous imprimerez le sujet pour lundi prochain (nous le lirons ensemble)**
  - **Essayez de former vos groupes de TP d'ici là (exactement 2 ou 3 membres par groupe)**
  - **Lisez le sujet attentivement et pensez-y !**

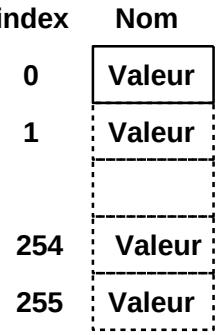
Rappels d'algorithmique

- **Nous avons vu :**
  - **Définition : Un algorithme est une suite finie d'instructions qui, étant donné des paramètres typés, calcule une solution d'un problème donné**
  - **Variables,**
  - **Constantes**
  - **Opérateurs, fonctions**
  - **Expressions**
  - **Une instruction : l'affectation**
  - **Une méthode de programmation utilisant tout ça : la programmation événementielle**



Rappels d'algorithmique

- **Notion de tableau :**
  - Collection de valeurs d'un même type
  - Accessibles par indexation
  - 1 ou plusieurs dimensions
  - De taille statique (fixe) ou dynamique
  - Généralement parcourus ou traités à l'aide de boucles



Notion de structure de contrôle

- **Branchement conditionnel :**

```
If condition Then
     [instructions]
Else
     [instructions]
End If
```

```
Select Case Expression
   Case ListeValeurs1
      [Instructions]
   Case ListeValeurs2
      [Instructions]
   Case Else
      [Instructions]
End Select
```


Notion de structure de contrôle

- **Itérations et itérations conditionnelles :**

```
For Compteur = Début To Fin [Step Incrément]
   [Instructions]
Next [Compteur]
```

```
Do While Condition
   Instructions
Loop
```

```
Do Until Condition
   Instructions
Loop
```

```
Do
   Instructions
Loop While Condition
```

```
Do
     Instructions
Loop Until Condition
```


Notion de structure de contrôle

Exemples d'utilisation de l'itération :

- Sommes : $\sum_{i=exp1}^{i=exp2} (exp(i))$

```
résulat = 0 'élément neutre pour l'addition
For i=exp1 To exp2
 résultat=résultat + exp(i)
Next i
```

- Produits : $\prod_{i=exp1}^{i=exp2} (exp(i))$

```
résulat = 1 'élément neutre pour la multiplication
For i=exp1 To exp2
 résultat = résultat * exp(i)
Next i
```

Exercice

- Écrivez la portion de code VB qui permet de calculer :

$$\sum_{i=1}^{i=20} \sum_{j=1}^{j=20} (i+j)$$

Solution

Procédures et fonctions

- **Procédures :**
  - Une procédure est un ensemble d'instructions qui participent à une même tâche.

  Syntaxe :

```
[Public/Private][Static] Sub nom([liste_arguments])
    déclarations
    instruction
End Sub
```

  - Exemple de procédure événementielle :

```
Private Sub cmdQuitter_Click()
     instructions
End Sub
```


Notion de procédure

- Exemple de procédure publique :

```
Public Sub SaisieNote()
    Do
          Note = InputBox("Tapez une note")
          'demande la note à l'utilisateur
    Loop Until Note>=0 And Note<=20
End Sub
```

- Pour utiliser cette procédure, il suffira de :
  - l'appeler par son nom : `SaisieNote`
  - Utiliser l'instruction `Call` :

```
…
Call SaisieNote 'appel la procédure de saisie
…
```

Notion de procédure

- Les arguments/paramètres: une procédure peut prendre des arguments (utiliser des données en entrée)
- Déclaration dans les parenthèses:

```
Private Sub nom (nom1 As type1, …)
   Instructions
End sub
```

séparateur

- À l'appel :
  `Call nom(expr_du_bon_type,…)`
- Dans le code de la procédure, nom1 à la valeur de `expr_du_bon_type`

Notion de procédure

Exemple de passage de paramètre :

```
Private Sub Affiche(Message As String)
    MsgBox(Message)
End Sub
```

À l'appel :

```
…
Dim news As String
News=« cool ! »
Affiche(«Ift-20403»)        ' affiche ift-20403
Affiche (News)              ' affiche cool !
Affiche News                ' affiche cool !
Call Affiche(«news»)        ' affiche news
Call Affiche News           'erreur compilation
…
```


Notion de procédure

- Le mot clé `Optional` permet de définir des arguments optionnels :
  - De type Variant (automatique)
  - Tous les arguments suivants seront optionnels
  - La fonction `IsMissing()` permet de tester si l'argument a été donné ou non
  - Exemple :

```
Function CalculTVA(MontantHT As Single, Optional
  Taux) As single
   If IsMissing(T) then Taux=20.6
   CalculTVA = MontantHT * (1+Taux/100)
End Function
```

Appels: `MontantTTC = CalculTVA 100`
Ou : `MontantTTC = CalculTVA 100, 5.5`

Notion de fonction

- Une fonction encapsule aussi un ensemble d'instructions, mais retourne une valeur (désignée par le nom même de la fonction).
- Cette valeur doit être affectée au nom de la fonction avant la fin du bloc d'instructions.
- Syntaxe :

```
Function nom(arg1 As type, …) As Type
   Instructions
   Nom = exp_du_bon_type
   Instructions
End Function
```

- Il faut préciser le type de la valeur retournée.

Notion de fonction

- **Exemple de fonction :**

```
Public Function Carré(x As Single) As Single
   Carré = x * x
End Function
```

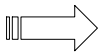
- **Exemples d'utilisation :**
  - `X=Carré(7)` `' assigne 49 à x`
  - `X=Carré(Carré(7))` `' assigne 2401 à x`
  - `X=Carré(2+Carré(7))` `' assigne 2601 à x`

Structure d'un projet

- **Un projet VB est fait de modules:**
  - **Modules de feuille (.frm)**
  - **Modules de code (.bas)**
- **Pour chaque module on a :**
  - **Une zone de déclaration de variables globales**
  - **Une zone de définition des procédures, les variables déclarées dans une procédure sont appelées variables locales**

Structure d'un projet

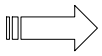
- **En VB, il y a quatre types de procédures :**
  - **Procédures évènementielles (juste dans les modules de feuilles)**
  - **Procédure VB pré-définies (`IIF`, `Inputbox`, …)**
  - **Procédures Sub**
  - **Procédures/fonctions `Function` : renvoies une valeur**



Tout le code exécutable VB doit être contenu dans des procédures

Appel / invocation de procédure

- **Différents types d'appel :**
  - `Call Nom-proc(argument1, …)`
  - `Nom-proc(arg1, …)` **ou** `Nom-proc arg1,…`
  - `Variable = NomFonct(argument1, …)`
- **Deux types de passages de paramètre à l'appel :**
  - **Passage par référence**
  - **Passage par valeur**



Distinction importante, commune à une majorité de langage de programmation haut niveau

Passage par référence

- C'est le mode par défaut:
  - C'est la variable qui est passée
  - Le nom de l'argument n'est alors qu'un alias (pseudonyme), c'est la variable qui est manipulée
  - Toute affectation sur le paramètre modifie la variable
  - Coté déclaration : rien ou `ByRef`
  - Coté appel : normal
    - `Nom(argument1, …)`
    - `Call Nom(argument1, …)`

Passage par valeur

- Seule la valeur de l'argument est passée (pas la variable)
- Donc seul le paramètre local peut être modifié, pas la variable
- Coté déclaration : `Byval`
- Coté appel : `nom((argument),…)`
  - On ajoute des parenthèses au paramètre passé

Exemple

```
Private sub CallVal1(Byval y As integer)
  y=y+1
End Sub
Private sub CallRef1(y As integer)
  y=y+1
End Sub
```

Appels :

```
Dim num As integer
Num = 2
CallVal1(num)  'après, num vaut toujours 2
CallRef1 num   'après, num vaut 3
CallRef1(num)  'après, num vaut toujours 3
CallRef1((num))'après, num vaut toujours 3
```


Les tableaux comme arguments

- Pour passer un tableau, il faut :

```
Call nomproc(nomTab())
Nomproc nomTab()
Nomproc nomTab(6)
Nomproc (nomTab(6))
```

Durée de vie et portée des identifiants

- Nous savons déclarer :
  - Variables : nom, type (taille), valeur
  - Procédures : procédures évènementielles, fonctions, …
- Chacun des identifiants déclaré a :
  - Une durée de vie : quand est-il défini ?, quand existe-t-il en mémoire ?
  - Une portée : où peut-on l'utiliser ?, où est-il visible ?

Durée de vie des identifiants

- Deux types de durée de vie pour les variables :
  - Automatique : par défaut en VB
    - crées au moment de leur déclaration et détruites à la sortie de la procédure (pour les variables locales), à la destruction du module (pour les variables globales).
  - Statique : crées grâce à `static`
    - Pour une variable : seules les variables locales peuvent être statiques
    - Procédure statique : toutes les variables locales sont statiques
    - Crées avec le module, détruites avec lui

Durée de vie des identifiants

- Exemple :

```
Private Sub Compte()
   Static dim compteur As Integer
   Compteur = compteur + 1
   Msgbox(«appel num:» & Str(compteur))
End Sub
```

- Appels:

```
Compte() 'affiche appel num:1
Compte() 'affiche appel num:2
 …
```


Portée des identifiants

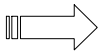
- La portée d'un identifiant désigne les régions du projet dans lesquels celui-ci est utilisable, visible, accessible
- Visible Þ vivant (pas le contraire)
- Exemple : lorsque l'on défini une variable locale dans une procédure, celle-ci n'est visible qu'entre sa déclaration et le `End Sub` (d'ailleurs, elle est détruite lorsque la procédure se termine, sauf si elle est statique)

Portée des identifiants

- **Trois types de portée en VB :**
  - **Portée locale (pour les variables locales): de la déclaration à la fin de la procédure**
  - **Portée de module (pour les variables globales et les procédures): accessible dans tous le module**
  - **Portée publique : grâce au mot réservé `Public`, on peut rendre les variables, procédures visibles des autres modules**

Portée des identifiants

- **Règle de visibilité : les identifiants locaux cachent les éventuelles identifiants de module ou publique.**
- **Même relation entre identifiants de module et identifiants publiques**



Important lors du travail en équipe. Sinon, attribuer deux fois un même identifiant est une mauvaise pratique.

Exemple

```
Private Sub one()
  Dim x As integer    'déclare x, variable locale
  x=26                'initialisation
  x=x+1               ' x est 27
End Sub
Private Sub two()
  x=x+1            'x doit être une variable globale
End Sub
```

Appel:

```
  Dim x As integer  'x est une variable locale ou globale
  X=6               'initialisation
  One()             'x est toujours 6
  Two()             'x est 7
```


Notion de récursivité

- **Puisque les procédures sont visibles partout dans leur module (et ailleurs si elles sont publiques), elles sont visibles dans leur propre code.**
- **Parfois, il est utile d'avoir des procédures qui s'appellent elle-même, directement ou indirectement.**
- **On parle alors de procédure récursive.**

Notion de récursivité

- Exemple de la factorielle:
- On a vu la version produit : n!= $\prod_{i=1}^{i=n}(i)$

```
Factorial = 1
For counter=number To 1 Step -1
  Factorial = Factorial * counter
Next counter
```

- Il existe aussi une définition récursive

$$\begin{cases} 0! = 1 \\ 1! = 1 \\ n! = n * (n-1)! \end{cases}$$

- 1ere solution (mauvaise):

```
Private Function Factorial (ByVal y As Double) As Double
   Factorial = y * Factorial(y-1)
End Function
```

- Faisons la trace d'exécution pour s'en convaincre:
  - Appel : Résultat = Factorial(5)

Notion de récursivité

- 2ème solution (bonne, testez là!)

```
Private Function Factorial(Byval y As Double) As Double
      If y <= 1 Then
         Factorial = 1                    'Cas de base
      Else
         Factorial = y * factorial(y-1)   'Pas récursif
      End if
End function
```

- Toujours bien spécifier le cas de base qui constitue la condition d'arrêt de la récursivité
- Importance des traces d'exécution

Notion de récursivité

- Exemple à éviter :

```
Sub Check1_Click ()
   If Check = 0 then
      Check1=1
   Else
      Check1=0
   End if
End Sub
```

- Cela a pour effet d'appeler indirectement (via l'événement `Click` sur la case à cocher), la procédure et ce de manière récursive.
- L'empilement des appels aura raison du système

Notion de récursivité

- **Autre exemple de fonction récursive:**
  - **La fonction d'Ackermann, `A:N*N->N` :**

```
A(m,n)=A(m-1,A(m,n-1)) si m>0 et n>0
A(m,0)=A(m-1,1) si m>0
A(0,n)= n+1 si n>0
```

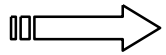
Exercice

1. **Se convaincre que la fonction est bien définie**
   `A(0,1) = 2`
   `A(1,1) = A(0,A(1,0))=A(0,A(0,1))=3`
2. **Écrire une fonction récursive qui la calcule**

Exercices

- **Écrire un programme qui tri les éléments d'un tableau indépendamment de sa taille:**
- **Traite les doublons**

index	TabEx
0	7
1	6
2	2
3	1
4	2



index	TabEx
0	1
1	2
2	2
3	6
4	7

Pour le prochain cours

- **Faire le tutoriel 5 (2ème partie du cours)**
- **Relire et annoter les transparents**
- **Imprimer et lire le sujet du TP1**
- **Former binôme ou trinôme**
- **Faire les exercices**

Questions