Introduction à l'algorithmique 2

DAKKAR Borhen-eddine

Lycée le Corbusier

BTS SN-IR

Table des matières

Table des matières

Exemple introductif

Nous charchons un algorithme qui lit 20 notes entières et qui indique le pourcentage des notes supérieures à 10.

Exemple introductif

Algorithme 1 : Pourcentage

```
Entier i   - - compteur de boucle sur les 20 notes
Entier nMoy   - - compteur du nombre de notes > 10
Entier note   - - pour lire une note
Réel pourcent   - - pourcentage de notes supérieures à 10
nMoy ← 0
Ecrire "donnez 20 notes entières"
Pour i de 1 à 20
    Lire note   - - lecture d'une note
    Si note > 10 alors
     nMoy ← nMoy + 1
Si nMoy = 0 alors
    Ecrire "aucune note supérieure à 10"
sinon
     pourcent ← (100.0 * nMoy)/20
    Ecrire "il y a ", pourcent, "% de notes > à 10"
Fin
```

Exemple introductif

Supposons que nous charshions en plus du pourcentage la moyenne des notes pour cette classe.
Avec la solution donnée précédemment, il est difficile de calculer la moyenne

parce que:

1. Nous n'avons pas enregistré les notes au moment de la lecture.
2. Il faut avoir accès à toutes les notes.

Nous pouvons suivre la même solution utilisée pour le calcul du pourcentantge.
Cependant, il existe une autre solution c'est d'utiliser les **tableaux**.

Définition d'un tableau

La notion des tableux est une notion très utilisée en algorithmique. Elle premet:

- De mnipuler plusieurs données en lui attribuant un seul nom.
- De repérer chaque valeur par ce nom et par son indice.

Pour l'exemple des notes, nous pouvons appeler notre tableax **Notes**. Avac Notes[1] la première valeur du tableau et Notes[20] la dernière valeur du tableaux.
Remarque : Il faut noter qu'un tableaux doit contenir le même type type de base (entier, réel, caractère, booléen).

Déclaration d'un tableaux

Pour déclarer un tableau, nous utilisons la syntaxe suivante :

Tab réel nom_tableau [taille]

Avec

- **Tab :** indique la déclaration d'un tableau.
- **réel :** le type des variables du tableau.
- **nom_tableau :** indique le nom du tableau.
- **taille :** indique le nombre d'éléments.

Revenons à notre exemple, nous pouvons déclarer un tableau de 20 valeurs de type réel comme suit :

Tab réel Notes [20]

Manipulation des éléments d'un tableau

- **Affectation :** Nous pouvons utiliser l'instruction d'affection $\leftarrow$ **(:=)** pour affecter des valeurs à notre tableau :

  Note[1] $\leftarrow$ 10
  Note[2] $\leftarrow$ 13
  Note[3] $\leftarrow$ 15

- **Lecture :** pour la lecture, nous pouvons utiliser l'instruction **Lire** comme suit : Lire Note[1]

- **Ecriture :** pour afficher le premir élément, nous pouvons utilisé l'instruction **Ecrire** comme suit: Ecrire Note[1]

Les instructions montrées ci-dessus manipulent les éléments du tableau élément par élément en utilisant un indice.

Manipulation des éléments d'un tableau

Il est possible d'utiliser de la boucle **Pour** permet la lecture et l'écriture des éléments d'un tableau.

- **Lecture :**

```
Pour i de 1 à 20 faire
    Lire Note[i]
Fin pour
```

- **Ecriture :**

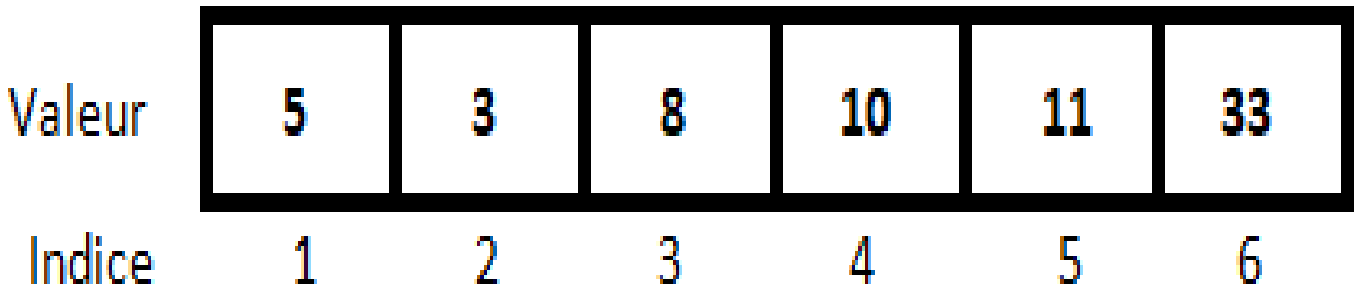
```
Pour i de 1 à 20 faire
    Ecrire Note[i]
Fin pour
```

Manipulation des éléments d'un tableau

- **Initialisation d'un tableau :** un tableau peut être initilisé avec l'instruction suivante:

  Tab réel T[6] $\leftarrow \{5, 3, 8, 10, 11, 33\}$

- Nous pouvons représenter ça par la figure ci-dessous:



Exemple

Algorithme 2 : Tableau

```
Tab entier c [6]
Entier i
Pour i de 1 à 6
    Lire c[i]
Fin pour
Pour i de 1 à 6
    c [i] ← c [i] * c [i]
Fin pour
Pour i de 1 à 3
    écrire c [i]
Fin pour
Pour i de 4 à 6
    écrire 2*c[i]
Fin pour
Fin
```

Exemple

- Lorsqu'on lui fournit en données les valeurs : 2, 5, 3, 10, 4 et 2.
- La première boucle place dans le tableau c les valeurs fournies en donnée, ce qui conduit à : 2, 5, 3, 10, 4, 2.
- La deuxième remplace ces valeurs par leur carré, soit : 4, 25, 9, 100, 16 et 4.
- La troisième affiche les trois premières valeurs, soit 4, 25 et 9.
- Enfin la dernière boucle affiche le double des trois dernières valeurs, soit 200, 32 et 8.

Table des matières

Les fonctions

- Il est pratique de pouvoir décomposer des parties relativement indépendantes d'un algorithme ce qui permet une compréhension facile et sans avoir à examiner l'ensemble du programme.
- Dans un algorithme, il est fréquent que l'on ait à réaliser en plusieurs endroits le même travail. Dans ces conditions, il serait regrettable d'avoir à réecrire des instructions identiques ou presque.
- On nomme **fonction** un ensemble d'instructions qu'on écrit une seule fois en leur attribuant un nom.
- Une fonction peut être utilisée ou appelée à tout point d'un algorithme .
- Une fonction peut être paramétrée, de façon que son travail puisse s'adapter à des situations semblables, mais non identiques.

Les fonctions

- Une fonction peut être déclarée comme suit:

```
fonction nom   - - en-tête
{
   Instructions   - - corp de la fonction
}
```

- Prenons l'exemple suivant:

```
Ecrire "Entrer la valeur de N"
Lire N
 ......
Ecrire "Entrer la valeur de N"
Lire N
```

Les fonctions

- Avec l'utilisation des fonctions, l'exemple précédent peut être écrit:

```
fonction lire_ecrire
{
    Ecrire "Entrer la valeur de N"
    Lire N
}
```

Fonction avec paramètre

- Nous pouvons définir une fonction avec un ou des paramètres (arguments).
- Regardons l'exemple suivant :
  fonction addition (entier a, entier b)
  { Entier somme
  somme = a + b
  Ecrire a, "+", b, "=", somme }
- La fonction **addition** permet de calculer la somme des deux entiers **a** et **b** et d'afficher le résultat.
- Il existe des fonctions qui peuvnet retourner des résultats. Si nous voulons retourner la **somme**, cette fonction devient :
  entier **fonction** addition (entier a, entier b)
  { Entier somme
  somme = a + b
  Ecrire a, "+", b, "=", somme
  Retourne somme }

Exemple

Algorithme 4 : programme utilisant la fonction max

```
entier n, p, q, m
n ← 3
p ← 5
q ← 2
m ← max (n, p, q)
Ecrire "maximum de", n, p, "et ", q, " = " , m
m ← max (2*n, p, q+2)
Ecrire "maximum de ", 2*n, p, "et ", q+2, " = ", m
entier fonction max (entier a, entier b, entier c)
{ Entier m
m ← a
Si b > m alors
    m ← b
    Fin si
Si c > m alors
    m ← c
Fin si
retourne m }
Fin
```

Table des matières

Transmission par valeur

- Prenons la fonction suivante:
  fonction incrémentation (entier i)
  {
  i $\leftarrow$ i+1
  .....
  }
- L'appel de cet fonction donne:
  incrémentation (n)
- Lors de l'appel de "incrémentation", la valeur fournie en paramètre a été recopiée dans un emplacement local à "incrémentation", nommé **i**.
- L'incrémentation de 1 de la valeur de **i** n'aura aucune incidence sur la valeur de la variable **n** du programme ayant appelé la fonction "incrémentation".
- On appelle ce mode de transmission des paramètres la transmision parvaleur.

Exemple transmission par valeur

Algorithme 5 : Echange valeurs de deux variables

```
Début
Entier n ← 10, p ← 20
Ecrire "avant appel : ", n, p
echange (n, p)
Ecrire "après appel : ", n, p
fonction échange (entier a, entier b)
{
  Entier c
  Ecrire "début échange : ", a, b
  c ← a
  a ← b
  b ← c
  Ecrire "fin échange : ", a, b
}
Fin
```

L'algorithme affiche:

avant appel : 10 20

début échange : 10 20

fin échange : 20 10

après appel : 10 20

Les échanges ont été opérés localement. Autrement dit aux vraibles de la fonction et ne concerneront pas les paramètres fournis lors de son appel.

Transmission par référence

- La transmission par référence (ou par adresse) est un mode de transmission dans lequel on transmet à la fonction l'adresse du paramètre effectif.

- La syntaxe de la transmission par référence est la suivante :
  fonction (référence type nom_variable)
  échange (référence entier a, référence entier b)

- En utilisant cette syntaxe, l'algorithme nous affiche maintenant :
  avant appel : 10 20
  début échange : 10 20
  fin échange : 20 10
  après appel : 20 10

- Avec ce mode de transmission, on peut noter que les valeurs des paramètres effectifs **n** et **p** ont bien été échangées.

La récursivité

- On parle de récursivité ou d'appels récursifs lorsqu'une fonction comporte un appel à elle même. Regardons l'emple suivant:

Algorithme 6 : Récursivité

```
Début
Entier n
Ecrire "donnez un entier positif : "
Lire n
Ecrire "Voici sa factorielle : ", fac(n)
- - la fonction fac
Entier fonction fac (entier n)
{
si n>1 alors
    retourne fac(n-1) * n
sinon retourne 1
}
Fin
```

- L'algorithme affiche:
  donnez un entier positif : 3
  Voici sa factorielle : 6

La récursivité

- L'appel de la méthode **fac** entraîne une allocation d'espace pour les éventuelles variables locales (ici, il n'y en a aucune), le paramètre n et le résultat.
- Chaque nouvel appel de **fac**, à l'intérieur de **fac**, provoque une telle allocation, sans que les emplacements précédents n'aient été libérés.
- Il y a donc une sorte d'empilement des espaces alloués aux informations gérées par la méthode, parallèlement à un empilement des appels de la méthode.
- Lors de l'exécution de la première instruction **retourne** que l'on commencera à « dépiler » les appels et les emplacements, donc à libérer de l'espace mémoire sur la pile [1].

Références

Delannoy Claude. *S'initier à la programmation et à l'orienté objet : avec des exemples en C, C++, C, Java, Python et PHP / Claude Delannoy*. fre. 2e édition. ISBN: 2-212-14067-3.