Structures de données Avancées et Algorithmique

Première Année ID & TDIA
Ecole Nationale des Sciences Appliquées – Al Hoceima

Prof A. Bahri
abahri@uae.ac.ma

A.U 2023/2024

Organisation du cours

- Rappel: Notions de base, types de base, opérateurs et expressions, les entrees et sorties, les instructions du contrôle, les tableaux, les fonctions, etc.
- La récursivité et paradigme « diviser pour régner »
- Les algorithmes de tri élémentaires et avancés
- Les structures de données élémentaires: liste, pile, arbre,etc.
- Les algorithmes de recherche
- Etc.

Rappel

Qu'est-ce que l'algorithmique ?

- Algorithme : mot dérivé du nom du mathématicien al_Khwarizmi qui a vécu au 9ème siécle.
- Un algorithme prend des données en entrée, exprime un traitement particulier et fournit des données en sortie.

Algorithme: Suite finie d'instructions vérifiant:

- Chaque étape est décrite de façon précise;
- Chaque étape est déterministe: produit des résultats uniques;
- L'algorithme s'arrête après un nb fini d'instructions
- Reçoit des données en entrée;
- Produit des données en sortie;
- Généralité: Applicable à des ensembles différents de données d'entrée

Qu'est-ce que l'algorithmique ?

- **Double problématique de l'algorithmique ?**
  1. **Trouver une méthode** de résolution (exacte ou approchée) du problème.
  2. Trouver une méthode **efficace**.

=>Savoir résoudre un problème est une chose, le résoudre efficacement en est une autre, ou encore montrer qu 'il est correcte ...!!

Instructions de base

– l'***affectation*** de variables
– la ***lecture/écriture***
– les ***tests***
– les ***boucles***

- un algorithme(prog) se ramène toujours à une combinaison de ces quatre petites briques.
  - ces briques ***opèrent*** sur des ***données*** (***variables***, ***constantes***) de ***différents types***.

Operateur d'affectation

Syntaxe :
variable <- donnée ;

Exemple :
variable
x,y :Entier ;
x<-5 ;
y<-x ;

Instruction de lecture

Permet de lire des valeurs à partir du clavier et les affecte aux variables.

La syntaxe de cette instruction est :
Lire (var1, var2, …);

Instruction de lecture : Exemple

•Lire(rayon);
A l'exécution de cette instruction, quand on saisit la valeur 8 au clavier elle sera la valeur de la variable rayon.
- lire les valeurs de plusieurs variables

Variables
nom : chaîne;
age : réel;
…
Lire(nom, age);
- nom et age sont des variables.

Instruction de lecture

- Remarque

Les arguments de Lire doit être des variables.

Contre-exemples
lire(3);
lire(x+y);

Instruction d'écriture

L'instruction d'écriture (écriture à l'écran) permet d'afficher à l'écran les valeurs des variables ou expressions après les avoir évaluées.

Sa syntaxe est la suivante:
écrire (expr1, expr2…);

Instruction d'écriture : Exemples

- Ecrire (rayon); affiche la valeur de rayon :5
- Ecrire (surface); affiche à l'écran la valeur de surface :78.5
- Ecrire (pi*rayon*rayon); affiche aussi 78.5
- Ecrire("surface"); affiche le mot surface.

Structure générale d'un algorithme

```
Algorithme Nom_algorithme;
Constantes
    Liste_de_constantes;
Variables
    Liste_de_varaibles;
Début
    Liste_instructions;
Fin.
```


Structure générale d'un algorithme : Exemple

```
Algorithme sommeN ;
Constantes
Variables
   n, somme: entiers ;
Début
   Lire(n) ;
   somme <-- n*(n+1)/2 ;
   Ecrire (somme); ou bien Ecrire("la sommes est:" ,somme) ;
Fin.
```


Condition et opérateurs

Les conditions sont exprimées par des opérateurs de comparaison et des opérateurs booléennes :
- Les opérateurs de comparaison sont =, <, <=, >, >= et ≠
- Les opéateurs boolénnes s'utilisent avec des opéandes booléns Et(And), ou (or) et non(not)

Définition de l'opérateur « ET »

Soit A et B deux expressions booléennes L'expression A ET B est vraie ssi A est vraie et B est vraie.

Exemples

```
Variables
test: booléen;
…
test <- (2>3) ET (2=2) ;
test <- (2<3) ET (2=2) ;
test <- (2>3) ET (2<2);
```


Définition de l'opérateur OU

Soit A et B deux expressions booléennes L'expression A ou B est fausse ssi A est fausse et B est fausse.

Exemples

```
Variables
test: booléen;
…
test <- (2>3) ou (2=2) ;
test<-(2<3) ou (2=2) ;
test<-(2>3) ou (2<2);
```


Définition de l'opérateur non

soit A est une expression booléenne. Si A est vraie, non(A) est fausse. Si A est fausse, non(A) est vraie.

Exemples
- test1 <- (1<2) ET (2=2) ;
- test2 <- non (non(1<2) ou (2=3));
- test3 <- non (test2);

Les structures de contrôles

- Une structure de contrôle sert à contrôler l'exécution d'une instruction ou d'un bloc d'instructions.
- Deux types de structures de contrôles:
  - Structure conditionnelle si l'exécution de l'instruction ou du bloc dépend d'une condition
  - Structure répétitive(itérative ou boucle) si l'exécution de l'instruction ou du bloc peut être répétée plusieurs fois(basée aussi sur une condition).

Instruction Si-Alors-Sinon

Si *p* **alors**
action

Si condition *p* vérifiée, exécuter *action*.

Si *p* **alors**
action 1;
Sinon
action 2;

Si condition *p* vérifiée, exécuter *action 1*. Sinon, exécuter *action 2*.

Si *x ≥ 0* **alors**
Début
x := x+1;
a := b+c;
Fin

Bloc de conditions entre **Début** et **Fin**.

Instruction Si-Alors-Sinon

- Exemple 1 :

```
Algorithme surfaceDisque;
Constantes
   PI=3.14;
Variables
   rayon, surface:réels ;
Début
   Lire(rayon);
  Si (rayon>0 ) alors
      Surface <- rayon*rayon*PI;
     Ecrire("surface=",surface);
 Finsi
Fin.
```


Instruction Si-Alors-Sinon

- Exemple 2:

Chercher la surface d'un disque de rayon saisi au clavier. Il faut s'assurer tout d'abord que le rayon soit positif. Si le rayon saisi est négatif, afficher le message "rayon non valide".

Instruction Si-Alors-Sinon

```
Algorithme surfaceDisque;
Constantes
   PI=3.14;
Variables
    rayon, surface:réels ;
Début
   Lire(rayon);
   Si (rayon>0 ) alors
      surface <-rayon*rayon*PI;
      Ecrire("surface=",surface);
   Sinon
      Ecrire(" rayon non valide");
  Finsi
Fin.
```


Structures conditionnelles : choix multiple

```
Selon (expression)
    Cas val1 : liste_instructions1;
    Cas val2 : liste_instructions2;
    . . .
   sinon : Liste_instructions;
Finselon
```

Structures conditionnelles : choix multiple

- Exemple

Ecrire un algorithme qui lit un opérateur op (+,-,/,*) et deux entiers a et b puis affiche le nom et le résultat de l'opération a op b.
Entrées op, a, b;
Sorties : la valeur de "a op b" et le nom de l'opération.

Structures conditionnelles : choix multiple

```
Algorithme operateur;
Variables
   op caractère;
   a,b, r : entiers;
Début
   Lire(op);
   Lire(a,b);
   Selon (op)
      Cas '+' : r<-a+b; ecrire ("la somme de a et b est:", r);
      Cas '-' : r<-a-b; ecrire("la différence entre a et b est:", r);
      Cas '*' :r<-a*b;ecrire("la multiplication de a par b est:", r);
      Cas '/' : r<-a/b; ecrire ("la division de a par b est:", r);
      Sinon : ecrire (op , " : opérateur non valide");
   Finselon;
Fin.
```


Instruction Tant que

Tant que *p* **Faire**
  action;

Tant que *p* est vrai exécuter *action*

Algorithme Plus-Grand-Element: Retourne la plus grande valeur d'une liste
  Entrée: *n* entiers S_1,..., S_n
  Sortie: *grand* contenant le plus grand élément

```
Début
   grand <- S1;
   i<-2;
   Tant que i ≤ n Faire
   Début
   Si Si > grand alors // une plus grande valeur a été trouvée
      grand <- Si;
   finsi
   i <- i+1;
   Fin
      Ecrire(« le plus grand est : »,grand)
Fin
```


Instruction Pour (For)

Pour *var <- init* **à** *limite* **Faire**
  action;

À chaque passage dans la boucle, *var* est incrémenté de 1. On s'arrête dès que *var > limite*

Algorithme Plus-Grand-Element: Réécriture de l'algorithme précédent mais avec une boucle "Pour"
  Entrée: *n* entiers S1,..., Sn
  Sortie: *grand* contenant le plus grand élément

```
Début
grand <- S1;
Pour i <-1 à n Faire
   Si Si > grand alors // une plus grande valeur a été trouvée
      grand <- Si;
   Finsi
   Ecrire(« le plus grand est : »,grand)
Fin.
```


Les tableaux

- Ensemble de données du même type
- Structure de données permettant d'effectuer un même traitement sur des données de même nature

tableau à **une** dimension

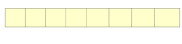
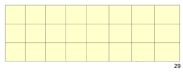


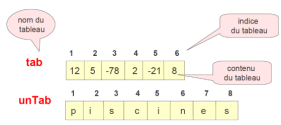
tableau à **deux** dimensions



Traitements opérant sur des tableaux

On veut pouvoir :
- **créer** des tableaux
- **ranger** des valeurs dans un tableau
- **récupérer, consulter** des valeurs rangées dans un tableau
- **rechercher** si une valeur est dans un tableau
- **mettre à jour** des valeurs dans un tableau
- **modifier** la façon dont les valeurs sont rangées dans un tableau (par exemple : les trier de différentes manières)
- effectuer des **opérations entre tableaux** : comparaison de tableaux, multiplication,...
- ...

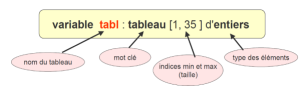
Définition du type



- **Remarques** :
- Indices : en général, démarrage à 1, **mais en C++, démarrage à 0**
-Nombre d'octets occupés : dépend du type des valeurs enregistrées

Déclaration d'un tableau

- Exemple : déclaration d'un tableau pouvant contenir jusqu'à 35 entiers



Définition d'un type tableau

- **type** <Nom> **=** <description>

- Exemple : *déclaration d'un nouveau type Mot, tableau de 10 caractères*

type Mot = tableau [1, 10] de **caractères**
variables nom, verbe : Mot

Utilisation d'un tableau : par les indices

- Accès en lecture :
- **afficher**(tabl[4]) *{le contenu du tableau à l'indice 4 est affiché à l' écran}*
- Accès en écriture :
  - tabl[3] ←18 *{la valeur 18 est placée dans le tableau à l'indice 3}*
  - **saisir**(tabl[5]) *{la valeur entrée par l'tilisateur est enregistrée dans le tableau à l'indice 5}*

  - attention!
    tab<-5 ; -->faux
    nom[2]= 10 ; -->faux

Tableaux à deux dimensions

- Déclaration
  Type points : **tableau**[1,2 ; 1,7] **d'entiers**

- Accès en lecture :
  - **afficher**(points[1,7]) *{la valeur contenue en ligne 1 colonne 7 est affichée à l'écran}*
- Accès en écriture :
  - points[2,4] ← 36
  - **saisir**(points[2,4]) *{la valeur fournie est enregistrée en ligne 2,colonne 4}*

Exemple : Saisir les valeurs d'un tableau 1D

```
Algorithme SaisieTableau {remplit un tableau avec nbVal valeurs entières }
constantes (TailleMAX : entier) ← 100
variables nbVal, ind : entier
    nombres : tableau [1, TailleMAX] d'entiers
début
   afficher ("Combien de valeurs sont à saisir?")
   saisir (nbVal)
   si nbVal > TailleMAX alors {refuser la saisie : la capacité du tableau est dépassée}
        afficher ("trop de valeurs à saisir")
   sinon
        pour ind ← 1 à nbVal faire
           afficher ("Donner une valeur") {valeur à ranger dans la indème case du tableau}
           saisir (nombres[ind])
        finpour
   fsi
fin
```

Procédures et fonctions

•Si vous avez un gros programme à mettre au point, et que certaines parties sont semblables, ou d'autres très complexes , alors il faut le structurer et chercher à utiliser au maximum "l'existant".

–utiliser une procédure (sous-programme) si un même traitement est effectué à plusieurs reprises dans le programme.

- Ä permet d'alléger le programme, de faciliter sa maintenance, de le structurer et d 'améliorer sa lisibilité.

Les procédures et les fonctions sont à la base de la programmation structurés

Procédures et fonctions

- Besoin de procédure et fonctions
  L'analyse d'un problème (démarche descendante) consiste à découper un problème complexes en tâches (sous-problème) que l'on est capable de définir.
  – Notion de tache : une tache est une action bien définie plus ou moins complexe qui s 'exerce à un instant donnée sur un ou plusieurs objets.
  – *Exemple* :
    - *Échanger* le contenu des variables entières A et B.
    - *Diviser* l 'entier A par l'entier B pour obtenir le quotient Q et le reste R
    - *Élever* A à la puissance B et mettre le résultat dans C
    - *Trier* la suite de N éléments contenu dans le tableau T
- …

Procédures et fonctions

- Déclaration :
  En pseudo :
  procédure nomProc(don_1:type,…,don_n:type, r rés_1,…,rés_m:type)
  début
  …
  fin

  fonction nomFonct(don_1:type,…,don_n:type) : typeValeurRetournée
  début
  …
  fin

Procédures et fonctions : exemples

```
procedure echanger(Entier a,b);
  Entier aux ;
Début
   aux:=a; a:=b; b:=aux;
fin;

fonction max(Entier a,b) :entier  ;
  Entier max ;
Début
     si(a>b) alors
          max<-a ;
     Sinon
          max<-b
     Finsi
     retourne max

Fin
```

Qu'est-ce que l'algorithmique ?

Pseudo-Code: Notation proche du code des langages de programmation (C, Pascal). Notation standard mais pas rigoureuse

- Titre de l'algorithme, description brève, entrées, sorties
- Procédures consécutives
- Numéroter les lignes (facultatif)
- Procédure commence par le mot ``**Procédure"**, suivit du nom, suivit des paramètres
- Instructions (lignes exécutables) entre ``**Début**" et ``**Fin**"
- Bloc d'instructions entre ``**Début**" et ``**Fin**"
- Ligne de commentaires signalée par // (ou /* */)
- ``**Retourne** (x)" termine une procédure et retourne la valeur de x

Exemple

Un algorithme de résolution de l'équation ax+b = 0

données : a et b entiers
Algorithme :
Écrire(' résolution de l 'équation : ax+b=0 ')
lire(a), lire(b)
Si a est non nul,
alors on obtient la solution : x = -b/a
sinon si b est non nul,
alors l'équation est insoluble
sinon tout réel est solution

- *Résultat : la solution de l 'équation ax+b=0; si elle existe*

Pseudo-Code

Algorithme maximum: Retourne le maximum de 3 entiers

- Entrée: Trois entiers a, b, c;
- Sortie: x contenant la valeur maximale parmi a, b, c

1. **Procédure** *max(a,b,c)*
2. *x := a;*
3. **Si** *b>x* **alors** // Si b plus grand que x, mettre x à jour
4. *x := b;*
5. **Si** *c>x* **alors** // Si c plus grand que x, mettre x à jour
6. *x := c;*
7. **Retourner** (x)
8. **Fin** max

Trace de l'algorithme pour:

a=1; b=5; c=3

x = 1
x = 5

Procédures et fonctions

- **Quand réaliser une fonction?**
  En général, le rôle d'une fonction est le même que celui d'une procédure.

  Il y a cependant quelques différences :
  - renvoi d'une valeur unique
  - la fonction est typé
    - Type de la fonction : scalaire, réel, intervalle, string,, structure
- **Exemple 1 :**
  F : R -> R
  x --> ax + b
  Incorporation dans une expression :
  if F(a0) > 10 then ...
- **Retour à l'exemple de tâches :**
  - ***Echanger*** le contenu des variables entières A et B.
  - ***Diviser*** l 'entier A par l'entier B pour obtenir le quotient Q et le reste R
  - ***Trier*** la suite de N éléments contenu dans le tableau T
  - ***Elever*** A à la puissance B

Procédures

Fonctions

Procédures et fonctions

Visibilité, localité et globalité des objets

- **Définitions** :
  - on appelle objet, une constante, une variable ou un paramètre formel.
  - On appelle objet local à une procédure, un objet défini dans cette procédure
  - on appelle objet global à une procédure P, un objet défini dans les procédures ancêtres de P (sur le chemin menant de P à la racine).
  - on appelle objet global à un programme, un objet défini au niveau 0.

Visibilité, localité et globalité des objets

- **Règle de visibilité des objets** :
  Dans une proc P de niveau n, les objets utilisables sont les objets globaux et les objets locaux à P.
  - Dans le cas où un objet local à une proc P, et un objet global à une proc P portent le même nom, l 'objet qui sera considéré est l 'objet local.
  - Plus généralement, dans le cas où deux objet globaux à la proc P portent le même nom, l 'objet qui sera considéré est l 'objet le plus proche de P (celui défini dans l 'ancêtre de P le plus proche).
- **Remarque :**
  les règles et définitions précédentes sont également valables dans le cas des fonctions.

55

56

Procédure récursive

- Procédure qui s'invoque elle-même

Exemple: $n! = n(n-1)(n-2)\ldots 2.1$ Si **n≥0**

$0! = 1$

$n! = n(n-1)!$ pour $n \geq 1$

5! = 5.4! 4! = 4.3! 3! = 3.2! 2! = 2.1! 1! = 1.0!

= 120 = 24 = 6 = 2 = 1

- Toute procédure récursive doit avoir une condition d'arrêt: cas de base qui permet de ne plus invoquer la procédure.

Ici, la condition d'arrêt est *n=0*

57

Entrée: Entier *n≥0*
Sortie: *n!*
Procédure *factoriel (n)*
Si *n = 0* **alors**
Retourne *(1)*
Retourne *(n. factoriel(n-1))*
Fin *factoriel*

factoriel(0):
Retourne: 1

Trace pour n =3:

factoriel(3):
Retourne: 3. *factoriel*(2)
3.2=6

factoriel(2):
Retourne: 2. *factoriel*(1)
2.1=2

factoriel(1):
Retourne: 1. *factoriel*(0)
1.1=1

58

Procédure itérative pour le calcul de n!

Entrée: Entier *n≥0*
Sortie: *n!*

Procédure *factoriel-iteratif (n)*
res = 1; courant = n;
Tant que *courant>0* **Faire**
*res := res * courant;*
courant := courant-1;
Fin Tant que
Retourne *(res)*
Fin *factoriel-iteratif*

Trace pour n =3 :

res = 1	*courant = 3*
res = 3	courant = 2
res = 6	courant = 1
res = 6	courant = 0

59

- Programme : série d'instructions pouvant s'exécuter en séquence, ou en parallèle (parallélisme matériel) qui réalise (implémente) un algorithme

60

61

62

63

64

65

66

67

68

69

70

71

72

73

74

75

76

77

78

79

80

81

82

83

84