ALGORITHMIQUE II

Pr. Mohammed AMMARI

Département d'Informatique

Faculté des Science -Rabat-

Université Mohammed V de Rabat

Remerciement : Pr. El Maati CHABBAR

PLAN DU COURS

OBJECTIF: **Conception des algorithmes corrects et efficaces**

PLAN

- RAPPELS : NOTATIONS ALGORITHMIQUES
- COMPLEXITE
- ALGORITHMES ITERATIFS DE TRIS
- RECURSIVITE
- DIVISER POUR RESOUDRE
- PREUVE D'ALGORITHMES

- Un type est un ensemble de valeurs sur lesquelles on définit des opérations.
  - Types de base :
    - ✓ **Entier** : Opérateurs arithmétiques +, -, *, div, mod
    - ✓ **Réel** : Opérateurs arithmétiques +, -, *, /
    - ✓ **Booléen** : Opérateurs logiques et, ou, non
    - ✓ **Caractère** : constante (lettre imprimable) entre apostrophe.
  - Les opérateurs relationnels permettant de faire des comparaisons: $<, \leq, =, >, \geq, \neq$

    Le Résultat de la comparaison est une valeur booléenne.

- Une **variable** possède :

  - un nom
  - une valeur
  - un type

(la valeur d'une variable peut changer au cours de l'exécution)

Déclaration : <variable> : <type>

- Une **expression**, pour un type, est soit une constante, soit une variable, soit constituée à l'aide de constantes, de variables, de parenthèses et des opérateurs

- INSTRUCTIONS
- **Affectation :** <variable> := <expression>

- **Condition :** **si** <condition> **alors**
  action
  fsi

ou :

- **si** <condition> **alors**
  action1
  sinon
  action 2
  fsi

(condition est une expression à valeur booléenne;
action est une instruction ou un bloc d'instructions séparées par ;)

- **Itération**

• boucle Pour

pour <variable> **:=** <initiale> **à** <final> **faire**
action
fpour

(Où initiale et finale sont des expressions de même type que celui de la variable contrôlant la boucle, le type peut être entier, caractère ou énuméré)

Remarque: la boucle **pour** affecte la valeur de initiale à variable et compare cette valeur à celle de finale avant d'exécuter action.

- Exemple: calcul de 1+2+…+n (n entier≥ 1 fixé)

```
Programme somme_des_n_premiersTermes
// partie déclaration
n : entier; s : entier; i : entier;  (ou n, i, s : entier)
Début
// Lecture des données
      Écrire (" n = ? " );  lire(n);
// calcul de la somme
      s := 0;
      pour i := 1 à n faire
                  s := s + i;
      fpour;
// affichage du résultat
      écrire("1+ 2 + … + n =  ", s);
fin
```

- en C

```
main ( ) {
// déclaration des variables
      int i, n, s;
// lecture des données
      prinf("   n = ?  "); scanf(" %d" , &n);
// calcul de la somme
      s = 0;
      for(i = 1; i <=n; i++)  s = s + i;
// affichage
      printf(" 1+2+…+ %d = %d \n" , n, s)
}
```

- **Itération**

• boucle tantque

tantque < condition> **faire**
action
ftantque

• boucle répéter

répéter
action
jusque < condition>

Remarques:

- Condition est une expression à valeur booléenne, cette expression doit être modifiée dans le corps de la boucle (dans action).
- La boucle pour peut être traduite en boucle tantque (ou en répéter) mais l'inverse n'est pas toujours vrai.

```
Programme somme_des_n_premiersTermes
// partie déclaration
n : entier; s : entier; i : entier;  (ou n, i, s : entier)
Début
// Lecture des données
      Écrire (" n = ? " );  lire(n);
// calcul de la somme
      s := 0;
      i := 1;
      tantque i <= n faire
                  s := s + i;
                  i := i + 1;
      ftantque;
// affichage du résultat
      écrire("1+ 2 + … + n =  ", s);
fin
```

- en C

```
main ( ) {
// déclaration des variables
      int i, n, s;
// lecture des données
      prinf("   n = ?  "); scanf(" %d" , &n);
// calcul de la somme
      s = 0;
      i = 1;
      while (i <= n) {
                  s = s + i;
                  i = i + 1;
      }
// affichage
      printf(" 1+2+…+ %d = %d \n" , n, s)
}
```

Algorithme

- Description formelle d'un procédé de calcul qui permet, à partir d'un ensemble de données, d'obtenir des résultats.
- Succession finie d'opérations

DONNEES algorithme **RESULTATS**

Cheminement à suivre :

Problème —analyse→ Algorithme —codification→ programme

Conception structurée des algorithmes

Découper l'algorithme en sous algorithmes plus simples. Chaque sous algorithme est un algorithme.

Fonctions

- Une fonction est un sous algorithme (sous programme) qui, à partir de données, retourne un seul type de résultat.
- Une fonction
  - possède un nom
  - communique avec l'extérieur par le biais des paramètres
  - retourne un résultat par l'instruction **retourner**(expression)

Schéma d'une fonction :

fonction <nom de la fonction>(Liste des paramètres) : <type du résultat>

// déclaration des variables locales

début

// corps de la fonction qui contient l'instruction retourner

fin

- Les paramètres de la définition d'une fonction (appelés formels) sont :
  - types
  - séparés par ',' s'il y en a plusieurs
- Les paramètres formels sont utilisés pour ne pas lire les données dans une fonction.
- Les variables déclarées dans une fonctions (y compris les paramètres formels) sont appelées variables locales.

- Exemple: factoriel n en pseudo code:

```
Fonction fact(n : entier) : entier
  m, i : entier;
début
    m  := 1;
    pour i :=2 à n faire
            m := m * i;
    fpour
    retourner(m);
fin
```

- n! en C :

```
unsigned  int fact (unsigned int n) {
  unsigned int  i, m;
  m = 1;
  for (i = 2; i<= n; i++)
      m = m * i;
  return m;
}
```

- L'**appel** d'une fonction est utilisé dans une instruction sous la forme :

  <nom de la fonction> (liste des paramètres effectifs)

- Les paramètres formels et effectifs doivent correspondre en nombre et en type
- Lors d'un appel :
  - les paramètres formels reçoivent les valeurs des paramètres effectifs correspondant
  - le corps de la fonction est exécuté jusqu'au premier retourner rencontré
  - l'exécution se poursuit (à l'dresse de retour) dans la fonction appelante.
- Remarque: le type de retour d'une fonction peut être vide, et dans ce cas on écrit retourner() ou pas d'instruction retourner;

- Exemple: calcul de 1+2+…+n (n entier ≥ 1 fixé)

```
Programme somme_des_n_premiersTermes
// partie déclaration
n : entier; r : entier;   (ou n, r : entier)
Début
// Lecture des données
    Écrire (" n = ? " );  lire(n);
// appel et utilisation de la fonction sommeArith
    r := sommeArith(n);
    // affichage du résultat
    écrire("1+ 2 + … + n =  ", r);
fin
```

- Fonction pour calculer la somme 1+2+…+m

```
Fonction sommeArith ( m : entier) : entier
i, s  : entier;
début
    s := 0;
    pour i := 1 à m faire
            s := s + i;
    fpour;
retourner(s);
fin
```

- **Deux types de passages des paramètres :**

• Passage par valeur: la fonction travaille sur une copie du paramètre effectif transmis; i.e. la valeur du paramètre effectif n'est pas modifiée après l'appel de la fonction.

• Passage par adresse (ou par référence):

- l'identificateur du paramètre formel est précédé par le mot **ref**.
- la fonction travaille directement sur l'identificateur du paramètre effectif; i.e. toute modification sur le formel entraine la même modification sur le paramètre effectif correspondant.

- Passage par valeur

```
Fonction échnger(x: réel, y : réel) : vide
z : réel;
début
        z := x;
        x := y;
        y := z;
fin

Fonction appelante()
a, b : réel;
début
        a := 2;  b:= 7;
        échanger(a,b);
        écrire( a = , a);
        écrire(  b = , b);
fin
```

Les résultas affichés par la fonction appelante :

a = 2 **b = 7**

- Passage par réference

```
Fonction echnger(ref x: réel,  ref y : réel) : vide
z : réel;
début
        z := x;
        x := y;
        y := z;
fin

Fonction appelante()
a, b : réel;
début
        a := 2;  b:= 7;
        échanger(ref a, ref b);
        écrire( a = , a);
        écrire(  b = , b);
fin
```

Les résultas affichés par la fonction appelante :

a = 7 **b = 2**

- Un **algorithme** se comporte comme une fonction sauf que l'on ne s'occupe pas des déclarations des variables ni de leurs types.

➢ Schéma d'un algorithme :

<nom de l'algorithme>(Liste des paramètres)

début
//bloc d'instructions
fin

Ou

Algorithme <nom de l'algorithme>
Données : // les variables qui sont des données de l'algo.
Résultat(s): // variable(s) contenant le(s) résultat(s)

début
//bloc d'instructions
//ne contenant pas retourner
fin

On omet la partie déclaration des variables locales en adoptant la règle: les variables simples sont en minuscule et les tableaux en majuscule.

- **Algorithme pour calculer n!**

```
factoriel(n)
// n ≥ 0
début
        m := 1; i := 1;
        tantque i < n faire
                i := i + 1;
                m := m * i;
        ftantque
retourner(m);
fin
```

TABLEAUX STATIQUES

- TABLEAUX
- Un tableau est utilisé pour représenter une suite d'éléments <u>de même type</u> ($T = (t_1,t_2,\ldots,t_n)$).
- Déclaration d'un tableau (à un dimension):

  <nom du tableau> : tableau [1 .. max] de <type des éléments>

  (exemple T : tableau [1..20] de réel)

  où : - max est une constante entière (positive)

  - le type des éléments est quelconque (de base ou déclaré).

- La taille (ou longueur) d'un tableau est le nombre d'éléments du tableau; elle est toujours inférieure ou égale à max.

- les tableaux, en algorithmique, commencent à l'indice 1(en C et en java, ils commencent à l'indice 0)

TABLEAUX STATIQUES

- Opérations (de base) sur les tableaux:
- Accès à un élément du tableau :

  <nom du tableau> [<indice>]

  (indice est une expression de type entier)
- Parcours : (on utilise un indice et une boucle pour ou tant que)

  (exemple :

  pour i=1 à n faire //n est le nombre d'éléments du tableau T

  traiter(T[i]) // traiter() est une fonction ou algorithme à définir

  fpour;
- Recherche d'un élément dans un tableau
- Insertion d'un élément
- Suppression d'un élément

TABLEAUX STATIQUES

- Exemple : recherche de la position d'un élément dans un tableau de réels.

```
fonction localiser(T: tableau[1..max]de réel, n : entier, val :réel) : entier
    i : entier;
    trouve : booléen;
    début
            i := 1; trouve := faux;
            tantque (i ≤ n) et (non trouve) faire
                    si (T[i] = val) alors
                        trouve := vrai;
                    sinon  i := i + 1;
                    fsi;
            ftantque;
             si (trouve) alors retourner(i)
            sinon retourner(0);
            fsi
    fin
```

- insertion d'élément x dans un tableau T à n éléments à la position p.

```
Fonction inserer(T : tableau[1..max] de réel, ref n : entier, x : réel, p : entier) : vide
// 1 ≤p ≤ n
i : entier;
début
    i := n;
    tantque i ≥ p faire
            T[i+1] := T[i];
            i := i – 1;
    ftantque
    T[p] := x;
    n := n + 1;
fin
```

- Suppression d'une valeur d'un tableau T, qui se trouve à la position p.

```
Fonction supprimer(T : tableau[1.. max] de réel, ref n : entier, p : entier) : vide
i : entier;
début
    i := p;
    tantque i < n faire
            T[i] := T[i+1];
            i := i + 1;
    ftantque;
    n := n – 1;
fin
```

- Remarques :

✓ Le nom du tableau, dans une liste de paramètres formels, est une référence.

(en C, l'adresse de T[i], notée T + i, est calculée:

adresse(T[i]) = adresse(T[0]) + sizeof(<type des éléments>) * i)

✓ L'insertion (resp. suppression) d'un élément à un indice i, nécessite un décalage des (n- i +1) derniers éléments d'une position à droite (resp. à gauche)

STRUCTURE (ENREGISTREMENT)

- Structures (ou enregistrements)
- Le type structure est utilisé pour représenter une suite d'éléments pas nécessairement de même type, chaque élément est appelé champs.
- Déclaration d'un type structure:

  <nom de type structure> = structure

  <variable_champs1> : <type_champs1>;

  <variable_champs2> : <type_champs2>;

  .

  .

  fstructure

- Déclaration d'une variable de type structure:

  <variable_structure> : <nom de type structure>

STRUCTURE (ENREGISTREMENT)

- Par analogie au type struct de C:
- Déclaration :

struct <nom de la structure> {
<type_champs1> <variable_champs1>
<type_champs2> <variable_champs2>
.
.
}

- Déclaration d'une variable de type structure:

struct <nom de la structure> <variable>

- Ou avec la définition de type : **typedef**

typedef struct <nom de la structure> <type_structure>;
Déclaration d'une variable :
<type_structure> <variable>

STRUCTURE (ENREGISTREMENT)

- Exemples :

```
Adresse = structure
        numero_rue : entier;
        nom_rue : tableau[1..20] de caractère;
        code_postal : entier;
fstructure

Point = structure
      abscisse : réel;
      ordonnee : réel;
fstructure

Déclaration des variables:
adr : Adresse;
p : Point
```

- En C :

```
struct adresse {
      int  numero_rue;
      char[20]  nom_rue;
      int  code_postal;
}
typedef struct adresse  Adresse;

typedef struct point {
      float  abscisse,  ordonnee;
}Point;

//Déclaration des variables:
Adresse  adr;
Point  p;
```

STRUCTURE (ENREGISTREMENT)

- Opération sur les structures :
- Accès à un champs :

  <variable de type structure>. <variable_champs>

  (ex: p.abcisse désigne l'abscisse du poin p)
- Affectation :

  <variable_type_structure> := <variable_type_structure>

  (L'affectation se fait champs par champs)

  Exemple : manipulation des complexes

STRUCTURE (ENREGISTREMENT)

- Déclaration de type complexe:

```
Complexe = structure
                p_reel, p_imag : réel;
fstructure

fonction plus(z1 : Complexe, z2 : Complexe) : Complexe
  z : Complexe;
debut
     z.p_reel  := z1.p_reel + z2.p_reel;
     z.p_imag  := z1.p_imag + z2.p_imag;
     retourner(z);
fin
```

- Déclaration de type complexe en C:

```
typedef struct complexe {
     double p_reel, p_imag ;
} Complexe;

Complexe  plus(Complexe z1,Complexe z2){
  Complexe  z;
   z.p_reel = z1.p_reel + z2.p_reel;
  z.p_imag = z1.p_imag + z2.p_imag;
  retun z;
}
```