Mise en œuvre d'algorithmes

KMIXIF21

L2-MIASHS

V. Camps – UPS/IRIT - camps@irit.fr

Organisation du module

- Objectif
  - Permettre aux étudiants d'acquérir les compétences algorithmiques et de programmation nécessaires à l'écriture de programmes en langage C
  - Décomposition d'un problème "complexe" en une suite organisée d'actions élémentaires qui pourra être ultérieurement traduite en langage C
  - Compréhension et vérification du comportement d'un algorithme (trace, tableau de situations)

Organisation du module

- Programme
  - Rappels d'algorithmique (variables et type, expressions et opérateurs, instructions, entrées/sorties, structures de contrôle, tableaux à une ou plusieurs dimensions)
  - Comprendre et réaliser la trace d'un algorithme (introduction à la complexité)
  - Introduction au langage C
  - Éléments de base du langage (structure d'un programme, syntaxe et éléments de base du C, entrées/sorties)
  - Structures de données simples (tableaux à une dimension, tableaux à plusieurs dimensions, chaînes de caractères, structures, …)
  - Fonctions (définition, passages de paramètres, utilisation)
  - Introduction aux pointeurs pour traiter le passage de paramètres par adresse

Notions générales

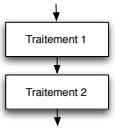
- Algorithmique
  - Faire réaliser par la machine une tâche complexe
  - Machine uniquement capable de réaliser des actions élémentaires (actions directement exécutables)
  - Solution : décomposer la tâche complexe en une suite organisée d'actions élémentaires → algorithme
- Algorithme
  - Description d'un comportement défini par une suite d'actions ou instructions (directives, ordres ou conseils) permettant d'atteindre un objectif fixé
    - Exemple : recette de cuisine, aide automobiliste
    - Si algorithme correct
      - Objectif atteint, résultat attendu obtenu
    - Si algorithme faux
      - Objectif non atteint, résultat obtenu aléatoire

Notions générales

- Propriétés d'un algorithme
  - Doit être exprimé en des termes compréhensibles par celui qui doit exécuter ces instructions
  - Doit être précis et non ambigu
  - Doit être exact
  - Aisément compréhensible
- Programme
  - Description du comportement de l'ordinateur pour résoudre un problème donné
  - Langage nécessaire
  - Algorithme = solution
- Langage nécessaire pour exprimer un algorithme
  - Indépendant des spécificités d'un langage de programmation
    - Pseudo code

Notions générales

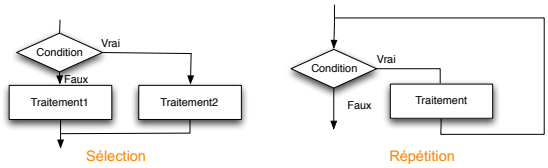
- Structures de contrôles
  - Organiser différentes actions élémentaires entre elles afin de construire un algorithme
  - Obtenir des actions composées (complexes, non élémentaires, …)
  - 3 structures de contrôle
    - Séquence : exprime l'enchaînement inconditionnel et séquentiel d'un ensemble d'actions (élémentaires ou composées)



Séquence

Notions générales

- Sélection : exprime une possibilité de choix entre deux actions (élémentaires ou composées) en fonction de la valeur de vérité d'une condition (vrai ou faux)
- Répétition : exprime le fait qu'une action (élémentaire ou composée) sera répétée tant que la valeur de vérité d'une condition sera vraie. La valeur de vérité de la condition doit être modifiée par l'action, sinon la répétition pourra être infinie



Notions générales

- ✓ Exemple
  - » Algorithme calculant le PGCD de deux entiers selon la méthode d'Euclide
    - o Méthode d'Euclide
      - – Si un des nombres est nul, l'autre est le PGCD sinon il faut soustraire le plus petit du plus grand et laisser le plus petit inchangé. Il s'agit ensuite de recommencer avec la nouvelle paire jusqu'à ce que un des deux nombres soit nul. Dans ce cas, l'autre nombre est le PGCD.
      - – Exemple: si a vaut 32 et b vaut 14, on obtient successivement

a	b
32	14
32-14=18	14
18-14=4	14
4	14-4=10
4	10-4=6
4	6-4=2
4-2=2	2
2	2-2=0

> → PGCD : 2

Notions générales

- » Algorithme calculant le PGCD de deux entiers selon la méthode d'Euclide

```
Début
   ENTIER a, b;
   AFFICHER "Saisir le 1er nombre: ";
   SAISIR a;
   AFFICHER "Saisir le 2nd nombre: ";
   SAISIR b;
   Tantque (non(a*b=0)) faire
      Si (a>b) alors
         a ← a-b;
      Sinon
         b ← b-a;
      Finsi
   Fintantque
   Si (a=0) alors
      AFFICHER "PGCD = ", b;
   Sinon
      AFFICHER "PGCD = ", a;
   Finsi
Fin
```


Notions générales

- ✓ Syntaxe
  - » Donne les règles d'écriture auxquelles se conformer pour écrire correctement un algorithme (les règles de grammaire…)
- ✓ Sémantique
  - » Décrit la signification de l'algorithme écrit en suivant les règles syntaxiques (elle exprime le sens de l'algorithme, ce qu'il veut dire…)
- ✓ L'algorithme
  - » Doit ensuite être traduit dans un langage de programmation (C, C++, Java, Scilab, AlgoBox…) pour être compréhensible et exécutable par un ordinateur

Variables

- » Les informations nécessaires au bon déroulement d'un programme informatique peuvent être
  - o Tapées au clavier (donc données par l' utilisateur)
  - o Issues de la mémoire
  - o Des résultats intermédiaires
  - o Des résultats définitifs
- » Ces informations sont stockées dans la mémoire de l'ordinateur sous forme de variables
- » Chaque variable est définie par un nom, une valeur, un type
- » Avant d'être utilisée, chaque variable doit être déclarée en lui assignant
  - o Un nom : significatif de son contenu
  - o Un type : caractérise les propriétés de la valeur mémorisée dans la variable
    - – Notation adoptée dans le cadre de ce cours : TYPE nom ← valeur;
- » Une variable possède une et une seule valeur à un instant donné qui peut évoluer dans le temps

Variables

- ✓ Déclaration de variables
  - » Identificateur
    - o Suite de caractères alphanumériques, le premier étant alphabétique
    - o Différenciation des lettres majuscules et des lettres minuscules
    - o Mots clés interdits
      - – 2essai est un identificateur invalide
      - – nom et NOM sont deux identificateurs différents
    - o Significatif de son contenu

Variables

- ✓ Types de variables
  - » Chaque variable est associée à un type définissant
    - o Un ensemble de valeurs possibles pour la variable
    - o Un ensemble d' opérateurs applicables sur ces valeurs
    - o 3 types principaux
      - – Numérique
        - > Notation : ENTIER ou REEL
      - – Alphanumérique pour caractère ou chaîne de caractères
        - > Notation : CARACTERE ou CHAINE
      - – Booléen pour les valeurs logiques VRAI et FAUX
        - > Notation : BOOLEEN

Variables

- » Affectation d'une valeur à une variable
  - o Opération qui permet d'assigner une valeur à une variable
  - o Notation : variable ← valeur;
  - o Exemple
    - – ENTIER som, nb;
    - – nb ← 5;
    - – som ← nb;
    - – som ← som + nb;

Type / Variable

- » Exemple

Type	Nom	Valeur
entier	somme	10
réel	moyenne	13,5
booléen	estTermine	faux
caractère	lettre	'a'
chaîne de caractères	nom	"dupond"

- » Traduction
  - o ENTIER somme ← 10;
  - o REEL moyenne ← 13.5;
  - o BOOLEEN estTermine ← FAUX;
  - o CARACTERE lettre ← 'a';
  - o CHAINE nom ← "dupond";

Expressions

- ✓ Constituées d'opérandes et d'opérateurs
  - » Opérateurs
    - o Unaires (un seul opérande est nécessaire)
      - – Placé immédiatement devant l'opérande sur lequel il s'applique
    - o Binaires (deux opérandes sont nécessaires)
      - – Placé entre les 2 opérandes sur lesquels il s'applique (notation infixée)
  - » Opérandes
    - o Peuvent être entiers (définis dans Z), réels (définis dans R) ou logiques (vrai ou faux)

Opérateurs

- ✓ Opérateurs
  - » Arithmétiques
    - o Unaire
      - – - (moins unaire)
        - > ENTIER som ← 1; som ← -som;
    - o Binaire
      - – + (addition)
        - > ENTIER a, b, s; a ← 1; b ← 2; s ← a + b;
      - – - (soustraction)
        - > ENTIER a, b, s; a ← 1; b ← 2; s ← a - b;
      - – * (multiplication)
        - > ENTIER a, b, s; a ← 1; b ← 2; s ← a * b;
      - – / (division réelle)
        - > ENTIER a, b; REEL s; a ← 1; b ← 2; s ← a / b;
      - – ^ (puissance entière)
        - > ENTIER a, b, s; a ← 2; b ← 3; s ← a ^ b;

Opérateurs

- – div (division entière)
  - > ENTIER a, b, s; a ← 3; b ← 2; s ← a div b;
- – mod (modulo)
  - > ENTIER a, b, s; a ← 1; b ← 2; s ← a mod b;
  - /*reste de la division entière de a par b*/

- » Logiques
  - o non
  - o et
  - o ou

a	b	non a	a et b	a ou b
V	V	F	V	V
V	F	F	F	V
F	V	V	F	V
F	F	V	F	F

Opérateurs

- Relationnels
  - = opérateur de test d'égalité
  - /= opérateur de test d'inégalité (différence)
  - <= opérateur d'infériorité ou d'égalité
  - < opérateur d'infériorité
  - > opérateur de supériorité
  - >= opérateur de supériorité ou d'égalité
- Priorité des opérateurs (décroissante)
  - non, - (unaire)
  - *, /, div, mod
  - +, -
  - <, <=, >=, >
  - =, /=
  - et
  - ou
  - Les opérateurs de même priorité sont évalués de la gauche vers la droite

Opérandes

- ✓ Désignent les informations sur lesquelles sont appliqués les opérateurs
- ✓ Un opérande peut avoir une des formes suivantes
  - Une valeur immédiate (littéral numérique ou logique)
    - 0 ou 3 ou 4.5 ou faux ou vrai
  - Une valeur nommée
    - PI
  - Une variable (l'information est le contenu de cette variable)
    - a, trouve, fini
  - Une expression (l'information est la valeur obtenue après évaluation de l'expression)
    - a + b
- ✓ Lorsqu'un opérande est une expression, cette expression est généralement parenthésée
  - a + (b / c)

Entrées/Sorties

L'ordinateur écrit les résultats à l'écran
Ces résultats sont lus par l'utilisateur

➔ Affichage écran

L'utilisateur écrit ses informations grâce au clavier
L'ordinateur lit ces informations et les stocke dans l'unité centrale

➔ Lecture clavier

Entrées/Sorties

- Affichage écran
  - AFFICHER s
    - Affiche l'expression (s) définissant la valeur à afficher
- Lecture clavier
  - SAISIR s;
    - Récupère la saisie au clavier et l'affecte à la variable s

```
Début
    /*…*/
    ENTIER nb;
    AFFICHER "Saisir une valeur";
    SAISIR nb;
    AFFICHER "La valeur saisie est egale a : ", nb;

    AFFICHER nb/10;
    /*…*/
Fin
```


Structures de contrôle - séquence

- ✓ Exprime l'enchaînement inconditionnel et séquentiel d'un ensemble d'actions (élémentaires ou composées)

Séquence

```
Début
    Action 1;
    Action 2;
    …
    Action n;
Fin
```

```
Début
  ENTIER nb;
  AFFICHER "Saisir la valeur de nb :";
  SAISIR nb;
  AFFICHER "La valeur de nb est egale a :", nb;
Fin
```

- Les mots Début et Fin sont des *mots-clés* précisant les limites de la séquence
- Un tel regroupement d'actions constitue une action composée
- Les actions composantes sont quelconques

Structures de contrôle - sélection

- ✓ Exprime une possibilité de choix entre deux actions (élémentaires ou composées) en fonction de la valeur de vérité d'une condition (vrai ou faux)

Sélection

```
Si (condition) alors
    Sequence1;
Sinon  /*partie sinon facultative*/
    Sequence2;
Finsi
```

```
Début
  /*…*/
  Si(delta < 0) alors
    AFFICHER "racines imaginaires";
  Sinon
    Si(delta = 0) alors
      x0 ← -b / (2*a);
    Sinon
      x1 ← (-b - rac(delta)) / (2*a);
      x2 ← (-b + rac(delta)) / (2*a);
    Finsi
  Finsi
  /*…*/
Fin
```

rac(x) désigne la racine carrée de x

Structures de contrôle - sélection

- Les mots Si, Finsi, alors et Sinon sont des *mots-clés*
- Les actions envisagées sont quelconques
- Une condition est une expression dont le résultat est une valeur logique (VRAI ou FAUX). Une telle expression doit être non ambiguë. Elle peut comporter des opérateurs de comparaison sur d'autres valeurs

✓ **Attention**

- Certaines formes simplifiées utilisées couramment ne sont pas autorisées dans les algorithmes. Si v doit être comprise entre MINIMUM et MAXIMUM, on ne peut pas écrire

  MINIMUM < v < MAXIMUM

  Il faut écrire

  (MINIMUM < v) et (v < MAXIMUM)

Structures de contrôle - sélection

✓ Exercice

- Ecrire un algorithme permettant de
  - lire au clavier deux valeurs entières (qui seront stockées dans deux variables distinctes `nb1` et `nb2`)
  - afficher à l'écran les valeurs respectives de ces variables
  - échanger la valeur de ces variables
  - afficher à l'écran les valeurs respectives de ces variables

✓ Exercice

- Écrire un algorithme permettant de
  - lire un nombre entier strictement positif
  - d'afficher à l'écran un message indiquant si ce nombre est ou pas un multiple de 3 et de 5

    On utilisera l'opérateur mod (en supposant a et b deux entiers strictement positifs, a mod b retourne le reste de la division entière de a par b)

Structures de contrôle - répétition

✓ Répétition en nombre inconnu

- Tantque, Fintantque, Répéter, Jusqu'à et faire sont des mots-clés
- Les actions répétées sont quelconques
- L'action est répétée tant que la condition produit la valeur logique vraie Cette condition est recalculée après chaque exécution de l'action répétée. Si la condition est fausse dès la première évaluation, l'action n'est pas exécutée (tant que)

Répétition

```
Tantque (condition) faire
    Sequence1;
    /*Mise a jour condition;*/
Fintantque

Répéter
    Sequence1;
    /*Mise a jour condition;*/
Jusqu'à (condition);
```

```
Début
   ENTIER i ← 1;
   Tantque (i<=3) faire
      AFFICHER "Bonjour !!!";
      i ← i + 1;
   Fintantque
Fin
```

```
Début
   ENTIER i ← 1;
   Répéter
      AFFICHER "Bonjour !!!";
      i ← i + 1;
   Jusqu'à (i>3);
Fin
```


Structures de contrôle - répétition

- Répétition en nombre connu
  - Les mots Pour, Finpour, dans et faire sont des mots-clés
  - Les actions répétées sont quelconques. Elles peuvent comprendre une seule action ou plusieurs actions qui seront exécutées en séquence. Parmi ces actions, peuvent se trouver d'autres répétitions ou tests

Répétition

```
Pour i dans borneInf..borneSup par pas de p faire
    Sequence1;
Finpour
```

```
Début
   ENTIER i;
   Pour i dans 1..3 faire
      AFFICHER "Bonjour !!!";
   Finpour
Fin
```


Structures de contrôle - répétition

- Lorsque l'ensemble de valeurs est défini par un intervalle de valeurs entières à parcourir de manière croissante ou décroissante, on pourra écrire

```
Pour i dans A..B faire
```

- ou, dans le cas décroissant

```
Pour i dans A.\.B faire
```


Structures de contrôle - répétition

✓ Exercices

- Écrire un algorithme permettant de calculer la somme des 10 premiers nombres entiers en utilisant la structure de contrôle répétitive appropriée puis d'afficher le résultat à l'écran

- Ecrire un algorithme permettant de
  - demander à l'utilisateur de donner le nombre de valeurs à traiter
  - lire chacune des n valeurs et les sommer si elles sont multiples de 3 et de 5
  - afficher la somme de toutes les valeurs saisies multiples de 3 et de 5

- Ecrire un algorithme permettant
  - de sommer les multiples de 3 et de 5 parmi une série d'entiers saisie au clavier. L'utilisateur est supposé entrer chaque entier un à un et terminer sa saisie en tapant sur 0
  - de calculer leur moyenne

Trace d'un algorithme

- ✓ Introduction aux tableaux de situations
  - » Situation
    - o L'ensemble des valeurs des variables à un instant donné du déroulement d'un algorithme
  - » Dans un algorithme, le résultat de l'évaluation d'une opération dépend de la valeur des variables qui sont les opérandes de cette opération
  - » Les seules opérations susceptibles de modifier la valeur des variables, et donc de modifier la valeur d'une situation, sont les opérations de lecture et d'affectation
  - » Avant l'exécution d'un algorithme, toutes les variables possèdent une valeur indéterminée

Trace d'un algorithme

- » A chaque algorithme on peut associer un ensemble de valeurs externes appelé ensemble des données
  - o L'algorithme prend cet ensemble en entrée en consultant les données une seule fois, et à tour de rôle
  - o On passe de la consultation d'une donnée à la consultation de la suivante dans l'ensemble des données par exécution d'une opération de lecture
- » Pendant l'exécution d'un algorithme, certaines erreurs peuvent provoquer un arrêt immédiat des opérations
  - o Division par zéro, exécution d'une opération de lecture alors que toutes les données ont déjà été lues, …

Trace d'un algorithme

- » Tableau de situations
  - o Permet de suivre pas à pas le déroulement d'un algorithme
  - o Permet de détecter les erreurs grossières des algorithmes
    - – Attention, ce n'est pas parce qu'un test d'un algorithme avec un tableau de situation particulier fournit les résultats attendus que cela prouvera qu'il fonctionne correctement pour n'importe quel cas !!!
    - – Par contre, un résultat erroné peut permettre de mettre en évidence des erreurs de conception
  - o Permet de pister la valeur des différentes variables et donc de déterminer de manière précise la faute commise pour la corriger
    - – On appelle ceci le déboguage d'un algorithme (ou, par extension, d'un programme sur ordinateur)

Trace d'un algorithme

```
Début
  ENTIER n, i;
  REEL r, q;

  SAISIR (n);              /* 1 */
  r ← 0;
  i ← 0;                   /* 2 */

  Tantque (i < n) faire
    i ← i + 1;
    q ← 1.0 / i;
    r ← r + q;             /* 3 */
  Fintantque

  AFFICHER ("r = ", r); /* 4 */
Fin
```

- o Donner le tableau de situations dans le cas où l'ensemble des données fourni est :
  - – { 4 }
  - – { 6 }
- o Que calcule et écrit cet algorithme ?
- o Modifier l'algorithme pour calculer la somme alternée

Trace d'un algorithme

- ✓ Exercice

```
Début
  ENTIER n, i ,a, b, c;

  SAISIR (n);
  i ← 1;                          /*1*/
  Tantque (i <= n) faire
    SAISIR(a) ;
    b ← 1 ;
    c ← 1 ;                       /*2*/
    Tantque (c <= a) faire
      b ← b * c ;
      c ← c + 1 ;                 /*3*/
    Fintantque
    AFFICHER ("b = ", b);         /*4*/
    i ← i + 1;
  Fintantque                      /*5*/
Fin
```

- o Donner le tableau de situations dans le cas où l'ensemble des données fourni est :
  - – {3, 2, 3, 4}
  - – {5, 5, 6, 2}
- o Que calcule et écrit cet algorithme ?

Complexité algorithmique

- ✓ Le calcul de la **complexité** d'un algorithme permet de mesurer sa **performance**
  - » 2 types de complexité
    - o **Complexité spatiale** : permet de quantifier l'utilisation de la **mémoire**
    - o **Complexité temporelle** : permet de quantifier la **vitesse** d'exécution
- ✓ Un algorithme ne doit pas seulement résoudre un problème
  - » Il doit être efficace
    - o Rapide (temps d'exécution)
    - o Econome en ressources (mémoire utilisées, espace de stockage)
      - – Besoin d'outils permettant d'évaluer la qualité théorique des algorithmes

Complexité algorithmique

» **Complexité algorithmique temporelle**
- Objectif : comparer l'**efficacité** d'algorithmes résolvant le même problème
  - Dans une situation donnée, établir lequel des algorithmes disponibles est le plus **optimal**
    - > Règles de ce calcul indépendantes
      - > du langage de programmation utilisé
      - > du processeur de l'ordinateur sur lequel sera exécuté le code
      - > de l'éventuel compilateur employé
- N'est pas mesuré en durée (heures, minutes, secondes)
- Utilisation d'unités de temps abstraites proportionnelles au nombre d'opérations effectuées
  - Chaque instruction basique (affectation d'une variable, comparaison, +, −, *, /, ...) consomme une unité de temps
  - Chaque itération de boucle rajoute le nombre d'unités de temps consommées dans le corps de cette boucle
  - Chaque appel de fonction rajoute le nombre d'unités de temps consommées dans cette fonction
    - > Nombre d'opérations effectuées par l'algorithme → addition

```
a ← b * 7;    → 1 multiplication + 1 affectation
              → 2 unités de temps
```


Complexité algorithmique

- Exemple
  - n! = n * (n-1) * (n-2) * ... * 2 * 1 (avec 0!=1)

```
ENTIER n, i, fact;
Début
   fact ← 1;                  initialisation : 1
   i ← 2;                     initialisation : 1
   Tantque (i <= n) faire     itérations : au plus n-1
     fact ← fact * i;         multiplication + affectation : 2
     i ← i + 1;               addition + affectation : 2
   Fintantque;
Fin
```

  - A chaque itération : 1 test
  - Nombre total d'opérations = 1 + 1 + (n - 1) * 5 = 5n − 3

  - Calcul pas exact
    - > On ne sait pas combien de fois la boucle va être exécutée
    - > Dans une condition, le nombre de comparaisons n'est pas toujours le même
      - > Test (a et b) : si a est faux inutile d'évaluer b

Complexité algorithmique

» La **complexité** en temps d'un algorithme exprimée par une fonction notée T dépend
- De la taille des données passées en paramètres : plus ces données seront volumineuses, plus il faudra d'opérations élémentaires pour les traiter. Soit n : nombre de données à traiter
- De la donnée en elle même, de la façon dont sont réparties les différentes valeurs qui la constituent
  - Ex : recherche séquentielle d'une valeur dans un tableau trié

- En toute rigueur, on peut distinguer deux formes de complexité en temps
  - La complexité dans le **meilleur des cas** : situation la plus favorable
    - > Ex : recherche d'un élément situé à la première position d'une liste
  - La complexité dans le **pire des cas** : situation la plus défavorable
    - > Ex : recherche d'un élément dans une liste alors qu'il n'y figure pas

- → On calculera le plus souvent la complexité dans le pire des cas, car elle est la plus pertinente
  - Mieux vaut toujours envisager le pire

Complexité algorithmique

» La complexité d'un algorithme mesure sa performance dans le "pire cas"
» Dans les situations où le problème prend le plus de temps de résolution
- L'algorithme ne prendre jamais plus de temps que celui estimé

» Pour comparer des algorithmes
- Pas nécessaire d'utiliser la fonction T
- Utiliser seulement l'ordre de grandeur asymptotique, noté O
- Une fonction $T(n)$ est en $O(f(n))$ si

$$\exists n_0 \in \mathbb{N}, \exists c \in \mathbb{R}^+, \forall n \in \mathbb{R}^+, n \geq n_0 \Rightarrow |T(n)| \leq c|f(n)|$$

$T(n)$ est en $O(f(n))$ s'il existe un seuil $n0$ à partir duquel la fonction T est toujours dominée par la fonction f, à une constante multiplicative fixée c près

Complexité algorithmique

- Approximation du temps de calcul : utilisation de la notation $O(.)$
  - Idée du temps d'exécution plutôt que calcul d'une expression précise inutilement compliquée
  - Utilisation des simplifications suivantes
    - > Mise à 1 des constantes multiplicatives
    - > Annulation des constantes additives
    - > Conservation des termes dominants
    - > Exemple
      - > Soit un algorithme effectuant g(n) = $4n^3 - 5n^2 + 2n + 3$ opérations
      - > On remplace les constantes multiplicatives par 1 : $1n^3 - 1n^2 + 1n + 3$
      - > On annule les constantes additives : $n^3 - n^2 + n + 0$
      - > On garde le terme de plus haut degré : n^3

$T_1(n) = 7 = O(1)$
$T_2(n) = 7n + 6 = O(n)$
$T_3(n) = 7n^2 + 6n + 4 = O(n^2)$
$T_4(n) = 7 + (n-1) * 6 = O(n)$

Complexité algorithmique

» Combinaison des complexités
- Instruction de base : temps constant noté $O(1)$
- Séquence
  - Addition des complexités d'opérations
    - > $O(f1(n)) + O(f2(n)) = O(f1(n) + f2(n))$

- Sélection
  - Addition des complexités d'opérations

```
Si (condition) alors          O(g(n))
   Séquence1;                 O(f1(n))
Sinon                         → O(g(n) + max(f1(n),f2(n))
   Séquence2;                 O(f2(n))
Finsi
```

- Répétition
  - Multiplication de la complexité du corps de la boucle par le nombre de répétitions

```
Tantque(condition) faire      O(g(n))
   Séquence1;                 O(f(n))   → O(m * (g(n) + f(n)))
Fintantque                                Si m itérations
```

Complexité algorithmique

- Calcul de la complexité d'un algorithme
  - Calcul de la complexité de chaque "partie" de l'algorithme
  - Combinaison des complexités
  - Simplifications

```
ENTIER n, i, fact;
Début
   fact ← 1;                  initialisation : O(1)
   i ← 2;                     initialisation : O(1)
   Tantque (i <= n) faire     (n-1) itérations : O(n)
      fact ← fact * i;        multiplication + affectation : O(1)
      i ← i + 1;              addition + affectation : O(1)
   Fintantque;
Fin                           → Complexité : O(n)
```


Complexité algorithmique

- Familles de complexité

Famille d'algorithmes	Notation	Exemples
Algorithmes **constants**	$\Theta(1)$	Echange deux valeurs
Algorithmes **logarithmiques**	$\Theta(\log n)$	Recherche binaire (dichotomique)
Algorithmes **linéaires**	$\Theta(n)$	Recherche séquentielle
Algorithmes **quasi-linéaires**	$\Theta(n \log n)$	Tri par fusion
Algorithmes **quadratiques**	$\Theta(n^2)$	Tri par sélection
Algorithmes **cubiques**	$\Theta(n^3)$	Produit de deux matrices
Algorithmes **polynomiaux**	$\Theta(n^p), p \geq 1$	
Algorithmes **exponentiels**	$\Theta(a^n), a > 1$	Calcul récursif de la suite de Fibonacci

- Choix entre plusieurs algorithmes
  - Situer leur complexité
  - Distinguer
    - Algorithmes polynomiaux ($O(n^k)$) → problème considéré facile
    - Algorithmes exponentiels dont la complexité ne peut être majorée par une fonction polynomiale → problème considéré "difficile"

Tableaux

- ✓ Un tableau à une dimension permet de stocker plusieurs valeurs, TOUTES étant de même type
- ✓ Chaque élément est contenu dans une case du tableau, indicée de 0 à la taille du tableau -1
  - Indice : peut être défini par une valeur, par une variable ou par une expression (l'expression est alors évaluée et sa valeur est utilisée pour accéder à l'élément du tableau)

```
constante ENTIER MAX ← 10;
ENTIER t[MAX];

t[0] ← 3;
t[MAX-1] ← 5;
```

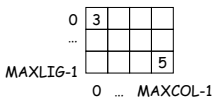
- ✓ Pour remplir un tableau
  - Saisir une à une chaque valeur
- ✓ Pour afficher le contenu d'un tableau
  - Afficher une à une chaque valeur
    - → Utilisation de la répétition

Tableaux

- ✓ Exercices
  - Ecrire un algorithme permettant de
    - remplir un tableau d'entiers comportant 5 cases; ces entiers seront saisis par l'utilisateur
    - afficher le contenu de ce tableau d'entiers
    - échanger le contenu des cases d'indice 1 et 3 du tableau
    - afficher le contenu de ce tableau d'entiers modifié
  - Ecrire un algorithme permettant
    - de lire N entiers saisis au clavier (N sera saisi par l'utilisateur) et de les stocker dans un tableau puis
    - De calculer la somme des N entiers saisis et de rechercher le minimum et le maximum parmi ces valeurs

Matrices

- Un tableau à deux dimensions (matrice) permet de stocker plusieurs valeurs, TOUTES étant de même type
- Exemple



- Chaque élément est contenu dans une case de la matrice, indicée par les numéros de ligne et de colonne à l'intersection desquelles elle est située

```
constante ENTIER MAXLIG ← 10;
constante ENTIER MAXCOL ← 10;
ENTIER m[MAXLIG][MAXCOL];

m[0][0] ← 3;
m[MAXLIG-1][MAXCOL-1] ← 5;
```

- Pour remplir une matrice
  - Saisir une à une chaque valeur (par ligne puis par colonne)
- Pour afficher le contenu d'une matrice
  - Afficher une à une chaque valeur (par ligne puis par colonne)
    - → Utilisation de la répétition (boucle imbriquée)

Matrices

- ✓ Exercices
  - Écrire un algorithme permettant
    - de définir une matrice d'entiers de dimension 5x7
    - de demander à l'utilisateur qu'il saisisse les valeurs de cette matrice
    - de demander à l'utilisateur qu'il saisisse un entier
    - de multiplier la matrice par cet entier puis
    - d'afficher la matrice résultat

Conventions d'écriture

- Un algorithme doit obéir aux conventions suivantes
  - Les mots du langage ne doivent pas servir de noms de variables; il s'agit de mots réservés ou "mots-clefs"
  - Choisir des noms de variables significatifs de leur rôle
  - Déclarer les variables et leur type
  - Indenter l'algorithme
  - Mettre une seule instruction par ligne
  - Rajouter des commentaires pertinents (non redondants)