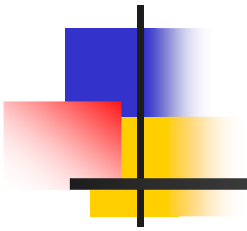


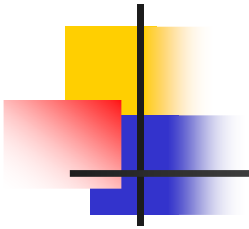
LIF1 : Algorithmique et Programmation C

Introduction à l'algorithmique

Coordonnées et site WEB

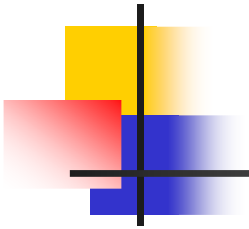
- Responsable de L'UE :
  - Elodie DESSEREE
  - Batiment Nautibus (2ème étage)
  - Tel : 04.72.44.81.92
  - Mél : elodie.desseree@liris.cnrs.fr
- Responsables d'amphis :
  - Jacques BONNEVILLE (séquence 4 à la Doua)
  - Elodie DESSEREE (séquence 5 à la Doua)
- Site WEB de l'UE (pour infos pratiques, supports, corrections, …)

  ➔ http://www710.univ-lyon1.fr/~edessere/LIF1/



Objectifs de la séance

- Replacer l'UE LIF1 dans son contexte Universitaire et définir ses objectifs.
- Les modifications dues au Plan Réussite en Licence
- Apprendre les rudiments du fonctionnement d'un ordinateur.
- Apprendre et manipuler le langage algorithmique.



Plan

- LIF1 : informations pratiques
- LIF1 / PCI / Autres UE informatiques
- Objectifs du module LIF1
- Fonctionnement interne d'un ordinateur
- Définition de l'algorithmique
- Syntaxe algorithmique
- Organisation de l'UE
- Environnement de travail

Le Plan Réussite en Licence

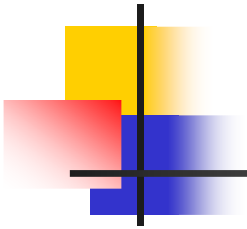
- Mise en place rentrée 2008
- Référent Pédagogique (~prof principal) qui suit votre évolution (2 rencontres dans le semestre)
- UE en 100% Contrôle Continu → pas de seconde session d'examen
- Calendrier adapté (contrôle terminal la semaine du 17 janvier)
- Heures d'enseignements supplémentaire (2 séances de TD de 1h30)
- Séances de soutien (sur avis du chargé de TD, de TP et du responsable d'UE) : 3 séances de 1h30 dans le semestre
- **Harmonisation** des notes en fin de semestre

Référents Pédagogiques

- 4 en informatique
  - Sylvain BRANDEL
  - Nathalie GUIN
  - Nicolas LOUVET
  - Florence ZARA
- 3 en mathématiques
  - Thomas BLOSSIER
  - Sébastien GAUTHIER
  - Dominique REYNAUD

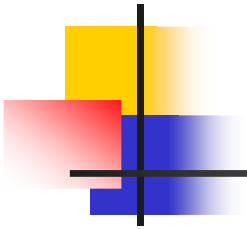
Modalité de Contrôle des Connaissances (MCC)

- En TD :
  - Un contrôle à chaque séance (11 séances)
    - Une question de cours (5 minutes)
    - 1 ou 2 exercices type TD (15-20 minutes)
  - Rendu des notes systématique la semaine suivante
- En TP : 4 notes (séances paires) un exercice noté en début de séance (15 min)
- Contrôle à mi-semestre
  - Épreuve de 1h sans document, non anonyme
  - Corrigée par les chargés de TD
  - mardi 2 novembre ou mercredi 3 novembre
- Contrôle terminal
  - Épreuve de 2h sans document, anonyme
  - Questions de cours, algorithmes, programmes C



Infos pratiques

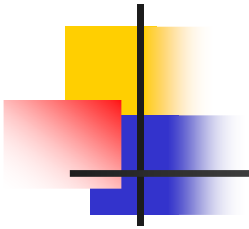
- Début des TDs : semaine du 20 septembre
  - 13 séances ➔ presque toutes les semaines
  - 2 séances la semaine prochaine
- Début des TPs : semaine du 27 septembre (8 séances ➔ regarder site pour voir planning)
- Contrôle à mi-semestre : 2 ou 3 novembre
- Contrôle terminal : semaine du 17/01/10
- Environnement de travail
  - Windows (cf PCI : TP1 environnement)
  - Répertoire utilisateur W:
- Outils complets :
  - Dev-Cpp (gratuit)
  - Microsoft Visual C++ (logiciel payant)



Autres UE informatiques

- Au L1
  - PCI : concepts informatiques généraux (bureautique, outil internet, réseaux, messagerie, création de pages WEB, …)
  - LIF2 : bases physiques de l'informatique
  - LIF3 : programmation fonctionnelle et récursive (Scheme), logique, réutilisation des notions vues en LIF1

- Au L2
  - LIF4 : base de données, réseaux
  - LIF5 : algorithmique / programmation avancés (suite LIF1)
  - LIF6 : architecture matérielle
  - LIF7 : conception et réalisation d'applications

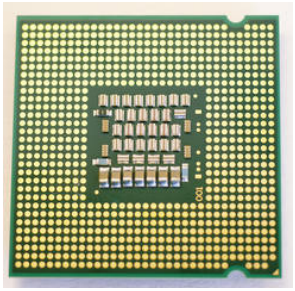


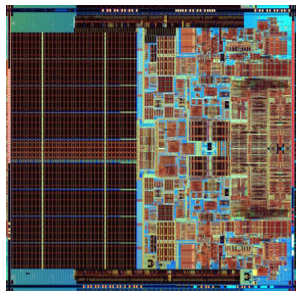
Objectifs de l'UE

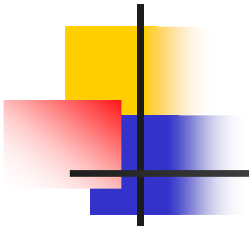
- Analyser un problème,
- Le formaliser
- Concevoir une solution (algorithme)
- Programmer l'algorithme
- Exécuter le programme sur un ordinateur

Composition d'un ordinateur

- Vision **simpliste** du contenu d'un ordinateur
  - Processeur : effectue les opérations
  - Mémoire(s), disques : stockage données, instructions
  - …
- Effectue des opérations à partir de données
- Vues d'un processeur







Pourquoi programmer ?

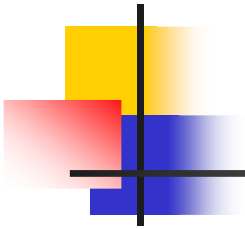
- Programmation existe partout
  - Magnétoscope
  - Réveil
  - Digicode …
- Besoin d'effectuer des nouvelles tâches → besoin d'écrire des programmes nouveaux
  - Par non informaticien : formalisation en français
  - Par informaticien : langage compréhensible par lui et la machine

Le processeur comprend :

- Programme (séquence d'instructions du processeur)

Code machine	Assembleur
cc2: 55	push %ebp
cc3: 89 e5	mov %esp,%ebp
cc5: 53	push %ebx
cc6: 83 ec 14	sub $0x14,%esp
cc9: e8 fc ff ff ff	call cca
cce: 81 c3 02 00 00 00	add $0x2,%ebx
cd4: 8b 45 08	mov 0x8(%ebp),%eax
cd7: 89 44 24 04	mov %eax,0x4(%esp)
cdb: 8b 45 08	mov 0x8(%ebp),%eax
cde: 89 04 24	mov %eax,(%esp)

- Seul langage compris par le processeur
- Codage hexadécimal des instructions → Quasi inutilisable pour programmeur



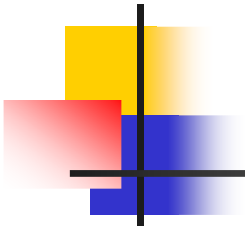
Le langage de programmation

- Langage commun entre
  - Le programmeur
  - Le processeur : traduit en assembleur puis en code machine
- Grande diversité
  - Langage C/C++ (LIF1, ce semestre)
  - Scheme (LIF3, prochain semestre)
  - Netlogo (PCI : TP 12)
  - Python, Java, Matlab, Mathematica, macros word / excel (écrites en Visual Basic for Applications VBA)…

Du problème au programme

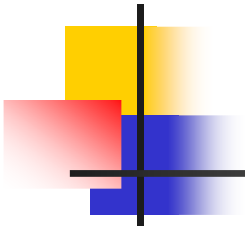
- Besoins exprimés en français (cahier des charges)
- Traduction dans un langage "universel" = algorithmique intermédiaire
- Traduction de l'algorithme en programme C
- Puis en code assembleur
- Puis en code machine compréhensible par le processeur

Non informaticien informaticien processeur



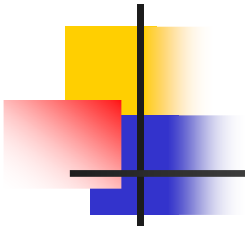
Algorithme : définition

- Un algorithme est une méthode
  - Suffisamment générale pour permettre de traiter toute une classe de problèmes
  - Combinant des opérations suffisamment simples pour être effectuées par une machine
- Pour un problème donné, il peut y avoir plusieurs algorithmes ou aucun
- Algorithmique : langage abstrait et non ambiguë



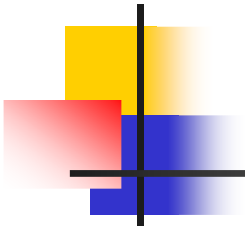
Algorithme : propriétés

- Solution correcte au problème à résoudre
- Explication aussi courte que possible
- Solution rapidement trouvée
  - ➔ Complexité en temps
- Solution utilisant un minimum de place mémoire
  - ➔ Complexité en espace



Algorithme : méthodologie

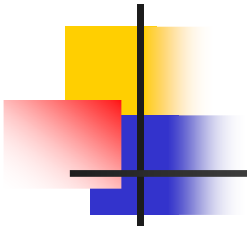
- Trois étapes caractérisent la résolution d'un problème
  - **comprendre la nature du problème** posé et préciser les **données** fournies ("entrées" ou "**input**" en anglais)
  - **préciser les résultats** que l'on désire obtenir ("sorties" ou "**output**" en anglais)
  - **déterminer le processus de transformation** des données en résultats.
- Ces trois étapes ne sont pas indépendantes et leur ordre peut être modifié.



Algorithmique / langage programmation

- Un algorithme est
  - Une **suite d'instructions élémentaires** décrites dans un langage universel exécutées de manière **séquentielle**
  - Indépendant du langage de programmation

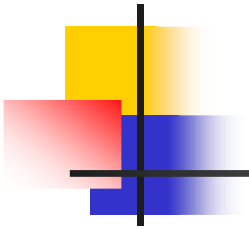
- Un langage de programmation
  - Est un langage commun entre machine et programmeur
  - Implante ou réalise un algorithme



La variable / la constante

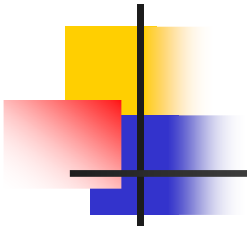
- Une **variable**
  - peut contenir un entier, un réel, un caractère…
  - associe un nom ou symbole à une valeur
  - sa valeur peut éventuellement varier au cours du temps
- Une **constante**
  - A une valeur fixée au début du programme qui ne change pas

    ➔ PI = 3.14159…



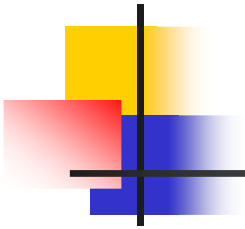
Le type des données

- une variable a un type caractérisant la valeur qu'elle contient
- Types utilisés en algorithmique :
  - Caractère : 'c' , 'a', '-', '!' …
  - Entier : 3 0 -3 -789
  - Réel : 0 3,345 -7,678
  - Booléen VRAI / FAUX
  - …



La déclaration des variables

- la *déclaration* permet de donner un nom à la variable
- éventuellement de lui associer un type, ainsi qu'une valeur initiale,
- Exemples
  - indice : entier permettra de déclarer une variable "indice" de type entier
  - Est_majuscule : booléen permettra de déclarer une variable booléenne
- La variable doit avoir un nom aussi évocateur que possible de son contenu



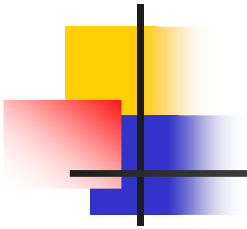
L'affectation

- Attribue une valeur à une **variable**
- Symbolisée en algorithmique par le symbole "←"
- La valeur peut être
  - le résultat d'une expression

    variable ← expression

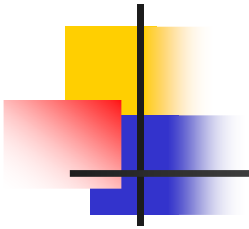
    var1 ← a + 2*racine(15)
  - Une valeur numérique

    a ← 2 (variable a contient le valeur 2)



Opérations sur les variables

- Affectation : variable ← expression
- La variable contient la valeur de l'expression
- Cette valeur est conservée jusqu'à la prochaine affectation
- Une variable peut apparaître dans une expression,
- elle sera remplacée par la valeur qu'elle contient au moment du calcul de l'expression



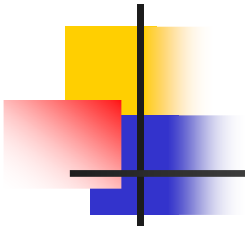
Contenu d'une variable

- Pour pouvoir stocker la valeur et vérifier qu'une variable est correctement utilisée,
- une variable a un type
- Un type est :
  - un domaine de valeurs (ensemble des valeurs possibles)
    - Entiers, réels
    - Booléen
    - caractères
  - un ensemble d'opérations pour manipuler ces valeurs
    - Addition, soustraction, multiplication, …
    - Opérations logiques
    - Concaténation, substitution, …

L'instruction, la séquence

- Instruction :
  - Opération élémentaire
  - Comprise et exécutée par le processeur
- Séquence :
  - Suite d'instructions
  - Délimitée par **Début** et **Fin (→ bloc)**

```
Début
        instruction1
        instruction2
        …
        instructionN
Fin
```



La conditionnelle

Si condition **alors**

instruction(s)

Sinon

instruction(s)

FinSi

- condition = expression booléenne (vrai/ faux)
  - Élémentaire
  - Complexe (conjonction, négation ou disjonction de conditions élémentaires et/ou complexes

La conditionnelle : exemples

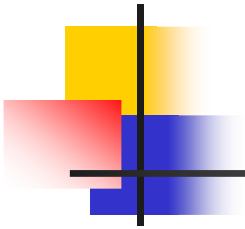
- Exemple 1 sans "sinon"

Si (A>2) **alors**
b←A*3
FinSi

- Exemple 2 avec "sinon"

Si ((A<10) **et** (B>racine(A*5))) **alors**
B←A*3
A← A+B
Sinon
A←A+2
B←A*B
FinSi

- Partie sinon facultative : il n'y a pas nécessairement de traitement à effectuer.
- Condition



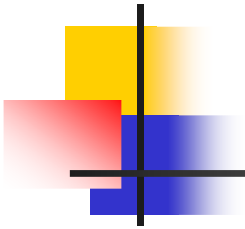
Itérative : boucle conditionnelle

- Permet de réitérer une instruction ou une suite d'instructions jusqu'à ce qu'une condition ne soit plus vraie
- Condition évaluée avant d'effectuer les instructions

TantQue condition **faire**

instruction(s)

FinTantQue



Boucle conditionnelle : exemple

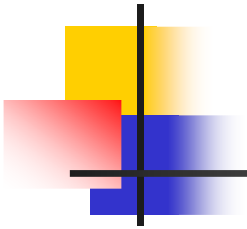
TantQue i<10 **faire**

a← a*i

i ← i+1

FinTantQue

- instruction qui modifie la condition pour éviter les boucles infinies



Boucle inconditionnelle : pour

- Cas particulier du TantQue

  Pour compteur **allant de** … **à** … **par pas de** … **faire**

  instruction(s)

  FinPour

- Permet de répéter un nombre connu de fois une suite d'instructions

Boucle inconditionnelle : exemples

- Compter de 1 à 10 (incrémentation)

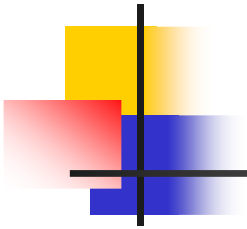
  a← 0
  Pour i **allant de** 1 **à** 10 par pas de 1 **faire**
  a← a + i
  FinPour

- Compter de 10 à 1 (décrémentation)

  a←0
  Pour i **allant de** 10 **à** 1 **par pas de -**1 **faire**
  a← a + i
  FinPour

- Compter de deux en deux

  a←0
  Pour i **allant de** 0 **à 10 par pas de** 2 **faire**
  a← a + i
  FinPour



Les entrées / sorties

- Assurent la communication programmeur / machine
- Données du problème (utilisateur → machine)

  Lire (valeur) ou **Saisir** (Valeur)
- Résultats affichés à l'écran (machine → utilisateur)

  Afficher (valeur) ou **Ecrire** (Valeur)

La condition

- Apparaît dans les "Tant Que"
- Variable booléenne qui renvoie comme valeur **VRAI** ou **FAUX**
- Combinaison de conditions : conjonction (**ET**), disjonction (**OU**), négation (**NON**)
- Tables de vérité

X	Y	X et Y
V	V	V
V	F	F
F	V	F
F	F	F

X	Y	X ou Y
V	V	V
V	F	V
F	V	V
F	F	F

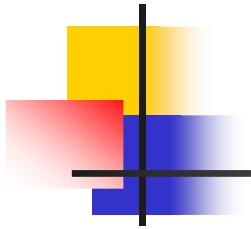
X	Non X
V	F
F	V

Exemple 1 : calcul du produit

- On veut calculer le produit de a par b et stocker le résultat dans une variable C
- En algorithmique on écrira :

a,b,c : réels	déclaration des variables
c ← a*b	stockage du résultat du calcul a*b dans la variable c

Exemple 2 : calcul du produit



- Dans cet algorithme, on n'utilisera pas la multiplication !!
- Raisonnement : $5 * 4 = \underbrace{5 + 5 + 5 + 5}_{4 \text{ fois}}$
- Généralisation : a * b = a+a+…+a (b fois)
- Formalisation : tant qu'on n'a pas ajouté **b** fois **a**, on ajoute **a** à la somme

Exemple 2 : calcul du produit

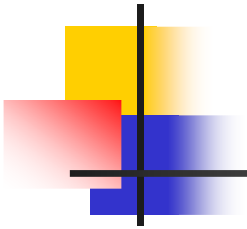
- Programme "complet" tel que vous aurez à les écrire dans le TD1.
- Traduction algorithmique avec un tant que

Début
a, b, somme : entiers
afficher ("donnez a et b")
lire (a)
lire (b)
somme← 0
TantQue b ≠ 0 **faire**
somme ← somme + a
b ← b - 1
FinTantQue
Afficher (somme)
Fin

Exemple 2 : calcul du produit

- Traduction algorithmique avec un "pour"

```
Début
    a, b, somme : entiers          /*mise en commentaires*/
    somme← 0                       /* initialisation de la somme à 0 */
    afficher ("donnez a et b")
    lire (a)
    lire (b)
    Pour i allant de 1 à b par pas de 1 faire
                    somme ← somme + a
    FinPour
    Afficher (somme)
Fin
```



Exemple 3 : minimum

- Détermination de la plus petite de 2 valeurs données par l'utilisateur
- Étapes de l'algorithme
  - Déclarer les variables à utiliser
  - Demander et saisir les valeurs de l'utilisateur
  - Comparer les deux valeurs
    - Utilisation d'une conditionnelle SI
  - Afficher la plus petite des deux

Exemple 3 : minimum

Début
  a,b : entier
  afficher ("donnez la valeur de a")
  lire (a)
  afficher ("donnez la valeur de b")
  lire (b)
  si (a<b) **alors** afficher (a "est la plus petite")
  sinon afficher (b "est la plus petite")
  fin si
Fin

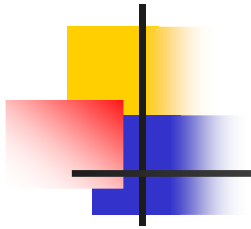
La partie "sinon" équivaut à la condition a ≥ b

Exemple 4 : minimum

Cette fois-ci on s'intéresse aussi au cas où les deux valeurs sont égales (ni a<b ni b<a) ➔ 2 "si" imbriqués !!!

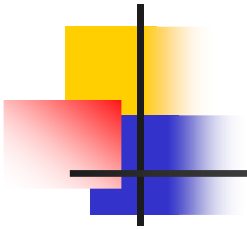
Début
a,b : entier
afficher ("donnez la valeur de a")
lire (a)
afficher ("donnez la valeur de b")
lire (b)
si (a<b) **alors** afficher (a "est la plus petite")
sinon si (b<a) **alors afficher** (b "est la plus petite")
sinon afficher ("les 2 valeurs sont égales")
fin si
fin si
Fin

La dernière partie "sinon" équivaut à la condition a = b



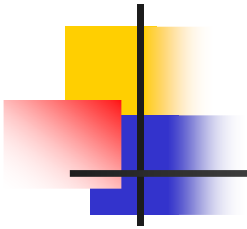
Exemple 5 : on continue l'imbrication

```
Si (val=1) alors afficher("Lundi")
    sinon si (val =2) alors afficher("Mardi")
        sinon si (val=3) alors afficher("Mercredi")
            sinon si (val=4) alors afficher("Jeudi")
                sinon si (val=5) alors afficher("Vendredi")
                    sinon si (val=6) alors afficher("Samedi")
                        sinon si (val=7) alors afficher("Dimanche")
                            sinon afficher("Erreur")
                        fin si
                    fin si
                fin si
            fin si
        fin si
    fin si
Fin si
```



Exemple 5 : Choix multiple

```
selon jour
    1 : afficher("Lundi")
    2 : afficher("Mardi")
    3 : afficher("Mercredi")
    4 : afficher("Jeudi")
    5 : afficher("Vendredi")
    6 : afficher("Samedi")
    7 : afficher("Dimanche")
   autrement : afficher("Erreur")
fin selon
```



Conclusion

- Tour d'horizon des notions de bases de l'algorithmique
  - Variable : déclaration, type
  - Instruction : séquence, bloc
  - Structures de contrôle
    - Conditionnelles : SI … ALORS … SINON …
    - Boucles : TANT QUE, POUR
- Exemples simples d'application