

INF 1141 - Algorithmique et programmation 1

LGI – Semestre 1

Département informatique

UFR des Sciences et technologies

Université de Thiès

Overview

1. PRÉSENTATION DU COURS
2. INTRODUCTION GÉNÉRALE
3. NOTIONS D'ALGORITHMIQUE
4. INTRODUCTION AU LANGAGE C
5. STRUCTURES DE CONTRÔLE
6. TYPES DE DONNÉES COMPOSÉS
7. POINTEURS

PRÉSENTATION DU COURS

Présentation générale

- **Unité d'Enseignement**
 - **Titre** : INFORMATIQUE
 - **Sigle** : INF 114
- **Élément constitutif**
 - **Titre** : Algorithmique et programmation 1
 - **Sigle** : INF 1141
- **Autres éléments constitutifs de l'UE (1)**
 - **Titre** : Introduction aux systèmes d'exploitation
 - **Sigle** : INF 1142

Volume horaire & Notation

- **CM : 30H**
- **TD/TP : 20H**
- **TPE : 50H**
- **Coefficient de l'UE : 4**
- **Crédits de l'UE : 10**
- **Evaluation**
 - **Contrôle des connaissances : 40%**
 - **Examen écrit : 60%**

Responsables

- **Magistral**

Pr Mouhamadou THIAM

Maître de conférences en Informatique

Intelligence Artificielle : Sémantique Web

Email : mthiam@univ-thies.sn

- **Travaux dirigés et pratiques**

M. X Y

Overview

1. PRÉSENTATION DU COURS
2. INTRODUCTION GÉNÉRALE
3. NOTIONS D'ALGORITHMIQUE
4. INTRODUCTION AU LANGAGE C
5. STRUCTURES DE CONTRÔLE
6. TYPES DE DONNÉES COMPOSÉS
7. POINTEURS

Chapitre 1

INTRODUCTION GENERALE

From scratch ...

Résoudre un problème

Indiquer où vous habitez



Décrire le cheminement permettant d'arriver à la solution d'un problème décrit par un énoncé

Décrire le cheminement à emprunter pour arriver à votre habitation telle que décrit par votre adresse

Quelques problèmes

- Un nombre N est-il divisible par 4 ?

- Soit une série de nombre, trier ces nombres

- Soit le problème classique de la tour de Hanoi, afficher la liste des mouvements nécessaires pour le résoudre.

- Un voyageur de commerce désire faire sa tournée, existe-t-il une tournée de moins de 50 km ?

David Hilbert & son problème n°10

- 1862 - 1943
- Liste des 23 problèmes de Hilbert (1900)
- Problème numéro 10 :
« Trouver un algorithme déterminant si une équation diophantienne à des solutions »



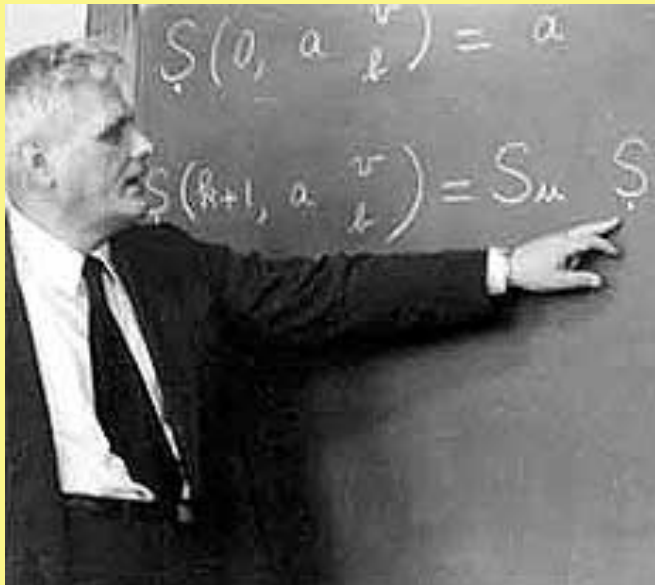
Équation diophantienne

- Une équation diophantienne, en mathématiques, est une équation dont les **coefficients** sont des **nombre entiers** et dont les solutions recherchées sont également **entières**. Le terme est aussi utilisé pour les équations à coefficients **rationnels**. Les questions de cette nature entrent dans une branche des mathématiques appelée **arithmétique**.

Équation diophantienne

- Carl Friedrich Gauss, un mathématicien du XIX^e siècle, disait : « Leur charme particulier vient de la simplicité des énoncés jointe à la difficulté des preuves »
- Identité de **Bézout** : $a x + b y = c$
- Théorème de **Wilson** : $(x - 1)! + 1 = y x$
- Triplet **pythagoricien** : $x^2 + y^2 = z^2$
- Last **Fermat** theorem ($n=4$) : $x^4 + y^4 = z^4$

Alonzo Church & le λ -calcul



- 1903 - 1995
- Résultat sur la calculabilité
- Développement du lambda-calcul
- 1936 : Démontre l'existence d'un problème indécidable
- Thèse de Church

Kurt Gödel



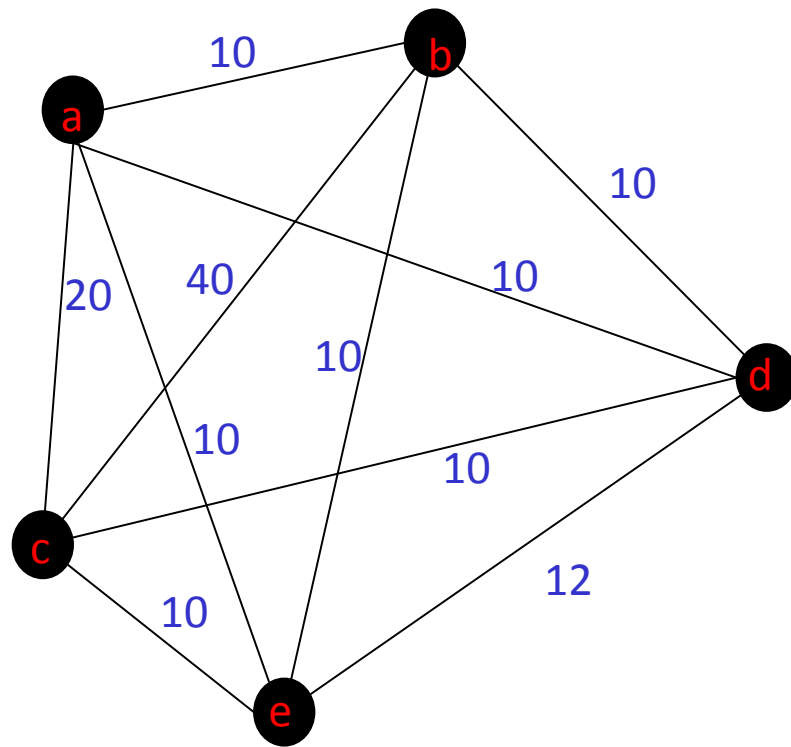
- 1906 - 1978
- Théorème d'Incomplétude

« pour tout système formel S contenant le langage de l'arithmétique, il existe une proposition G indémontrable dans S »

Exemple de problème de NP

Le voyageur de commerce

- Un voyageur de commerce désire faire sa tournée, existe-t-il une tournée de moins de 50 km ?



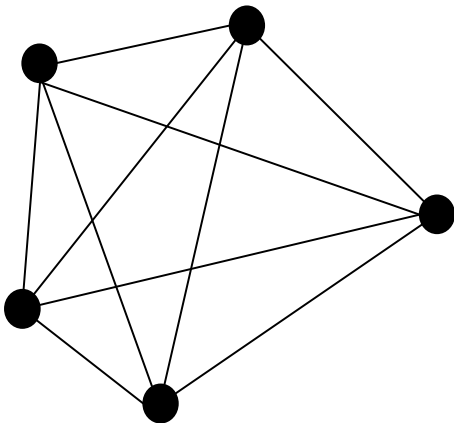
a-b-e-c-d-a

50km

NP et co-NP ?

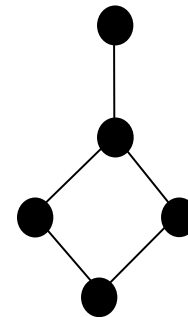
Problème du circuit hamiltonien

Un graphe est-il hamiltonien ?



Ce problème \in NP

Un graphe n'est-il pas hamiltonien ?



Ce problème \in co-NP

Est-ce que ce problème \in NP ?

WANTED

$$P \stackrel{?}{=} NP$$

Overview

1. PRÉSENTATION DU COURS
2. INTRODUCTION GÉNÉRALE
- 3. NOTIONS D'ALGORITHMIQUE**
4. INTRODUCTION AU LANGAGE C
5. STRUCTURES DE CONTRÔLE
6. TYPES DE DONNÉES COMPOSÉS
7. POINTEURS

Chapitre 2

NOTIONS D'ALGORITHMIQUE

1. Intuition
2. Historique
3. Variables
4. Expressions
5. Instructions
6. Écriture d'algorithme

Intuitivement ...

- **Loi de Greer:**

Un programme informatique ne fait jamais ce que vous voudriez qu'il fasse mais ce que vous lui dites de faire

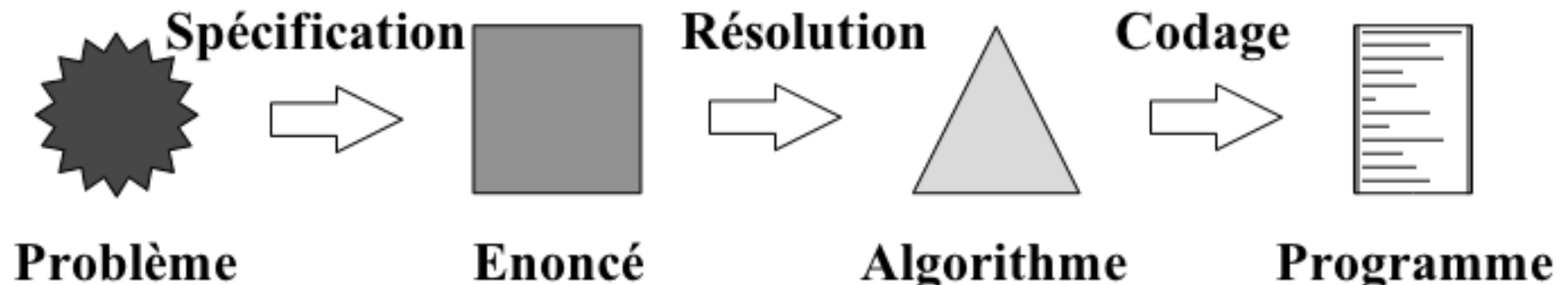
- **Pour rigoler – ne rigolez pas !**

Nous serons trop forts quand nous saurons que l'ordinateur est trop bête : elle ne sait faire que ce que nous savons lui expliquer comment on le fait.

Step by step ...

- **Notions de programme**

- Spécification d'un schéma de calcul sous forme d'une suite finie d'opérations élémentaires obéissant à un enchaînement déterminé



Ne pas se laisser aveugler par l'objectif final : le codage !

Programmer c'est quoi ?

- Communiquer avec
 - La machine
 - Soi-même
 - Les autres
- Pour réussir il faut des
 - Désignations évocatrices
 - Algorithmes en pseudo-code concis et clairs
 - Programmes indentés et commentés

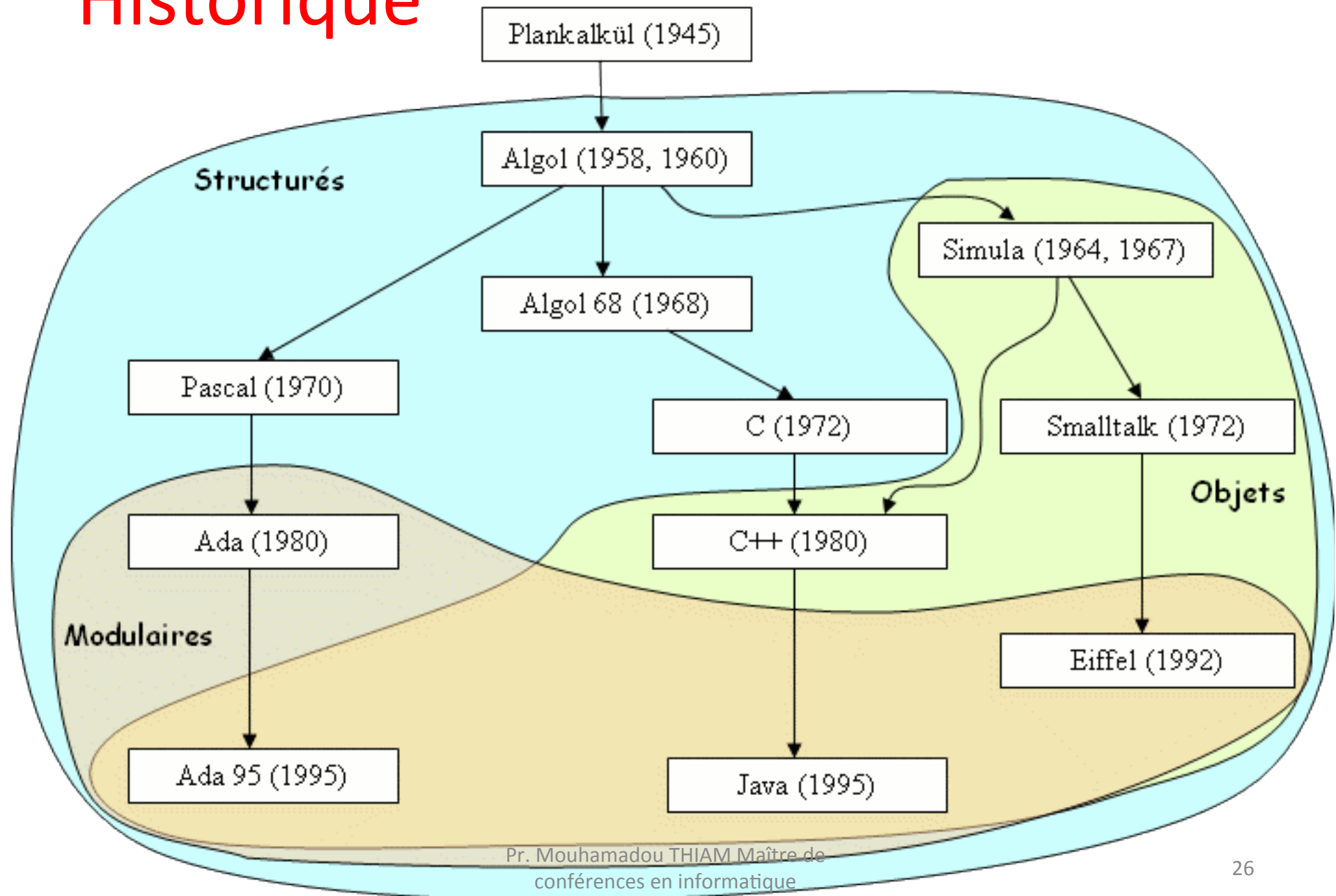
Programmation impérative

- Un paradigme de programmation qui décrit les opérations en termes de séquences d'instructions exécutées par l'ordinateur pour modifier les données.
- Principales instructions impératives
 - l'assignation
 - le branchement conditionnel
 - le branchement sans condition
 - le bouclage.

Un langage universel

- Un langage facilite la résolution de classes de problème
 - C : systèmes d'exploitation (Unix)...
 - C++ : applications de grande taille...
 - JAVA : applications de grande taille...
 - LISP : prototypage, systèmes experts...

Historique



Définitions (1)

- **Algorithme**
- Spécification d'un schéma de calcul sous forme d'une suite finie d'opérations élémentaires obéissant à un enchaînement déterminé
- La description des étapes à suivre pour réaliser une tâche (un travail).

Définitions (2)

- Langage de programmation
- **Compilé** (C, C++, Java, Pascal, ...) le programme source → compilateur
La compilation → programme exécutable
- **Interprété** (Perl, Lisp, Prolog...) le programme source a besoin, pour chaque exécution, d'un programme annexe appelé interpréteur

Définitions (3)

- Une **structure de données** est une façon d'organiser des informations dans la mémoire de l'ordinateur pour faciliter leur manipulation. Il existe plusieurs structures de données: ***tableau, structure, liste chaînée, arbre...*** Leur choix influe fortement sur la simplicité du programme et sur son efficacité

Programme vs algorithme

- Un **programme** est donc la description d'un *algorithme* dans un langage accepté par la machine
- Un **algorithme**, à l'inverse d'un *programme*, est indépendant du langage de programmation (et donc de ce machine)

Le langage algorithmique

- Spécialisé
 - pas de poisson,
 - Ni de viande, ...
- De haut niveau
 - pas de détails techniques
- Concis
- Modulaire

VARIABLES

Variables (1)

- Un algorithme manipule des données stockées dans des variables
- Chaque variable possède:
 - Un nom
 - Emplacement mémoire
 - Une adresse
 - Un type

Variables (2)

- Types de données de base
 - Entiers (12, 789)
 - Caractères ('a', 'g', '3')
 - Réels (-7.9; 0,5; pi; 1/3)
 - Chaînes de caractères ("je mange du ... ")
 - Booléens (vrai, faux)

Déclaration de variables

- **Déclaration** d'une variable = réservation d'un emplacement mémoire, auquel on donne un nom unique appelé ***identificateur*** et par lequel on peut accéder à sa valeur.
- **Syntaxe** :
 identificateur : type
- **Exemple**
 a : entier
 x, y, z : entier

Constantes

- **Une constante** est, comme une variable, un emplacement de la mémoire mais sa valeur ne peut pas changer au cours de l'exécution du programme

Déclaration de constantes

- **Déclaration** d'une constante est toujours associée à son initialisation (première valeur).

Syntaxe :

identificateur : type = valeur

- **Exemple**

Max : entier = 32767

Opérations de l'algorithmique

Type	Exemples de valeurs	opérations possibles	opérateur ou mot clé correspondant
réel	-15.69 , 0.36	addition, soustraction, multiplication, division, comparaison	+, -, *, /, <, ≤, >, ≥, =, ≠
entier	-10 , 0 , 3 , 689	addition, soustraction, multiplication, division, modulo, comparaison	+, -, *, div, mod, <, ≤, >, ≥, =, ≠
caractère	'B' , 'h' , '£' , '?'	successeur, prédécesseur, ordre, caractère, comparaison	suc, pred, ord, car, <, ≤, >, ≥, =, ≠
chaîne	"Bonjour" , "93000", "toto@caramail.com"	longueur, concaténation, comparaison	longueur , + , <, ≤, >, ≥, =, ≠
booléen	VRAI , FAUX	négation, conjonction , disjonction	NON , ET , OU

Remarques (1)

- Entiers, la division est notée **div** → chiffre avant la virgule du résultat (elle renvoie un entier).
- Les entiers supportent une opération supplémentaire appelée **modulo**, notée **mod** → reste de la division entière.
- Exemple:
 - 7 **div** 2 donne 3
 - 7 **mod** 2 donne 1

Remarques (2)

- Une valeur de type caractère doit être entre apostrophes (sinon le compilateur pourrait la confondre avec un nom de variable) : **'a'**
- Une valeur de type chaîne doit être entre guillemets (sinon le compilateur pourrait la confondre avec un nom de variable) : **"aa"**

Comparaison de caractères (1)

- Les caractères sont comparés selon l'ordre du code ASCII. Par exemple le caractère 'Z' (majuscule), de code ASCII 90 est inférieure au caractère 'a' (minuscule) de code ASCII 97
- Soit la variable c de type caractère:
 - **succ(c)** → caractère suivant c selon le code ASCII
 - **pred(c)** → caractère précédant c selon le code ASCII
 - **ord(c)** → code ASCII du caractère c.
 - **car(n)** → caractère correspondant au code ASCII n.

Comparaison de caractères (2)

- **Exemple**
 - `ord('A')` vaut 65
 - `succ('A')` vaut 'B'
 - `pred('A')` vaut '@'
 - `car(65)` donne 'A'
- L'opérateur `+` sert à concaténer 2 chaînes
- **Exemple:** `"bonjour"+"Zin-Zin"="bonjour Z-Z"`
- **Longueur**("`Zin-Zin`") = 7

Le type booléen

- 2 valeurs possibles: **VRAI** et **FAUX**
- Opérateurs de base : ET, OU, NON, OUEX
- Table de vérité

A	B	A ET B	A OU B	NON A
VRAI	VRAI	VRAI	VRAI	FAUX
VRAI	FAUX	FAUX	VRAI	FAUX
FAUX	VRAI	FAUX	VRAI	VRAI
FAUX	FAUX	FAUX	FAUX	VRAI

EXPRESSIONS

Expressions : Définition

- Une **expression** est une:
 - **variable** ou une **constante**
 - **Valeur**
 - exemple : "bonjour", 45
 - **combinaison de variables, de constantes, de valeurs et d'opérateurs**
 - exemple : $2 * r * 3.14$

Expressions : Evaluation

- Elle est évaluée **durant l'exécution** du programme, et possède une **valeur** (son évaluation) et un **type**.
- Par exemple, supposons que *rayon* soit une variable de valeur 5. Alors l'expression
 $2 * \textit{rayon} * 3.14 = 31.4$

Expression conditionnelle

- Appellation
 - expression conditionnelle
 - expression logique
 - expression booléenne
 - condition
 - expression dont la valeur est soit **VRAI** soit **FAUX**
- Possible affecter expression logique à variable
- Plusieurs types d'expressions conditionnelles

Les conditions simples

- Une condition simple est une comparaison de deux expressions de même type
- **Exemples**
 - $a < 0$
 - $op == 's'$
 - $(a + b - 3) * c == (5 * y - 2) / 3$
- Caractères → basée sur le code ASCII
 - $'a' < 'b'$ et $'s' > 'm'$

Les conditions complexes

- Formées de plusieurs conditions simples ou variables booléennes reliées entre elles par les opérateurs logiques **ET**, **OU**, **NON**
- Exemples
 - $(a < 0) \text{ ET } (b < 0)$
 - $((a + 3 == b) \text{ ET } (c < 0)) \text{ OU } ((a == c * 2) \text{ ET } (b \neq c))$
- Parenthèses \rightarrow régler d'éventuels problèmes de priorité des opérateurs logiques.

INSTRUCTIONS

Affectation

- Mettre une valeur dans une variable
- Représentée par \leftarrow
 - Variable \leftarrow expression
 - $X \leftarrow 12$
 - $X \leftarrow 5 + Y$

Affectation (suite)

- A gauche de \leftarrow nom de variable
- A droite de \leftarrow expression
- 2 étapes dans son évaluation
 - Déterminer la valeur de l'expression
 - Mettre le résultat dans la variable
- Pas d'affectation pour
 - Expression
 - Constante

Affectation (Fin)

- Possibilité de retrouver la variable à droite et à gauche de \leftarrow
 - $a \leftarrow a + 3$ est autorisé
 - Evalue $a + 3$ avec l'ancienne valeur de a
 - range le résultat dans a .
 - La valeur de a sera donc augmentée de 3

Lecture

- permet au programme de lire des données entrées au clavier
- affecte les valeurs entrées au clavier à des variables
- **Syntaxe**
 - Lire ($variable_1, variable_2, \dots, variable_n$)
 - Lire (x)
 - Lire (x, y)

Affichage

- Afficher des expressions à l'écran
- **Syntaxe**
 - **Ecrire**(expression₁, expression₂, ..., expression_n)
- **Exemples**
 - Écrire ("Bonjour!") → **Bonjour!**
 - Ecrire (x) → **Daba** si **x="Daba"**
 - Ecrire (x , "est agée de 19 " , y) → **Daba est agée de 19 ans** si **y= "ans"**;

ECRITURE D'ALGORITHME

Encore des théories ...

- *«Le vrai problème n'est pas de savoir si les machines pensent, mais de savoir si les hommes pensent » - B.F. Skinner*
- *« La question de savoir si un ordinateur peut penser n'est pas plus intéressante que celle de savoir si un sous-marin peut nager » - Edgar W. Dijkstra*

Écriture d'un algorithme

- Nom pour chaque algorithme (réutilisation)
- Variables d'entrée (données s'il y en)
- Variables de sortie (résultats)
- Variables locales
- Préciser leurs types

Structure générale d'un algorithme

Algorithme *nom de l'algorithme*

Type

Définition de types

Constante

Déclaration de constantes

Variable

Déclaration de variables

Définition de sous programmes

Début

instructions

Fin

Remarques

1. On peut insérer des commentaires
 - soit entre les balises `/* */`
 - soit après `//` jusqu'à la fin de la ligne.
2. Les types de base font partie du langage et n'ont pas à être définis par le programmeur

Exemple : afficher un message

Algorithme bonjour

//simple écriture d'un message de bienvenue, ne retourne rien

Début

Ecrire ("bonjour jespereke sa va bien")

Fin

Exemple : échanger 2 entiers

Algorithme Echange

variable

x , y : entier

tmp : entier

Début

Ecrire("Donner la valeur de l'entier x:")

Lire(x)

Ecrire("Donner la valeur de l'entier y:")

Lire(y)

Ecrire("Avant échange: x vaut ",x, "et y vaut ",y)

tmp \leftarrow x

x \leftarrow y

y \leftarrow tmp

Ecrire("Après échange: x vaut ",x, "et y vaut ",y)

Fin

Overview

1. PRÉSENTATION DU COURS
2. INTRODUCTION GÉNÉRALE
3. NOTIONS D'ALGORITHMIQUE
- 4. INTRODUCTION AU LANGAGE C**
5. STRUCTURES DE CONTRÔLE
6. TYPES DE DONNÉES COMPOSÉS
7. POINTEURS

Chapitre 3

INTRODUCTION AU LANGAGE C

Overview

1. Introduction
2. Types primitifs
3. Déclaration de variables et de constantes
4. Opérateurs du langage
5. Instructions élémentaires
6. Structures de contrôle
7. Structures de répétition
8. Instructions de rupture de séquence en C

Introduction (1/2)

- Le langage C a été conçu
 - en **1972**
 - par **Dennis Richie** et **Ken Thompson**
 - But : développer un système d'exploitation UNIX
- **1978**, *The C Programming language* par **Brian Kernighan** et **Dennis Richie**

Introduction (2/2)

- 1980, Apparition de compilateurs C
- 1983, Normalisation par l'ANSI (American National Standards Institute)
- 1989, définition de la norme ANSI C
- 1990, Reprise par l'ISO (International Standards Organization)

Principes

- Langage typé
- Langage compilé
 - Texte écrit en langage C
- Compilation
 - Edition de liens
 - Programme exécutable

Compilation / Exécution

- **Compilateur**
 - lit un fichier source
 - produit un fichier objet
- **linkers** : opération par laquelle plusieurs fichiers objets sont mis ensemble pour se compléter mutuellement

Structure d'un programme C

- Directives de compilation
- Définition structures et des types (s'il y en a)
- Déclaration variables globales (s'il y en a)
- Définition sous programmes (fonctions) (s'il y en a)
- programme principal (fonction main)

Structure d'une fonction

- Toutes les fonctions ont la même structure :
 - type de retour
 - nom de la fonction
 - liste d'arguments mis entre parenthèses et séparés par des virgules
 - le corps de la fonction contenu entre accolades.

Exemple de fonction

```
type_de_retour nom_prog(arguments)
{
    /* déclaration des variables utilisées dans
    la fonction main */
    /* instructions du programme*/
}
```


Programme principal : *main*

- Nom du programme principal = mot **main**
- Type de retour est en général **int** (le type entier en langage C).
- En général pas d'arguments. On met le mot **void** entre les parenthèses (ou rien du tout).
- Dans certains cas il peut prendre des arguments.

Corps du programme principal

```
#include <stdio.h>

int main( )
{
    printf("Bonjour");
    return 0;
}
```

Remarques

- **#include <stdio.h>** directive de compilation
→ des fonctions de la bibliothèque standard *stdio.h*
(*STandarD Input/Output*)
 - **Bibliothèque** (ou **librairie**) = fichier dans lequel est défini un ensemble de fonctions prêtes à l'emploi
- La fonction **printf** est définie dans **stdio.h**

La directive `#include`

- Indique au compilateur d'inclure des librairies
- Le compilateur C fournit un ensemble de librairies mais le programmeur peut aussi créer ses propres librairies.
- Syntaxe:
`#include <fichier>` ou `#include "fichier"`
- Exemple
`stdio.h, string.h, stdlib.h, ctype.h`

La directive #define

- Indique au compilateur de remplacer une chaîne par une autre
- permet une meilleure lisibilité du programme source, évite de devoir réécrire à chaque fois une suite longue de caractères .

- Syntaxe:

#define *nom_à_remplacer suite_de_caractères*

- Exemple: #define VRAI 1

Compilation, **VRAI** remplacé par le chiffre **1**

TYPES PRIMITIFS

Les entiers

Type	Description	Taille en octets	Valeurs possibles	Format
int	Entier standard	2 ou 4	de -32768 à 32767 (sur 2 octets) de -2147483648 à 2147483647 (sur 4 octets)	%d
short	Entier court	2	de -32768 à 32767	%d
long	Entier long	4	de -2147483648 à 2147483647	%ld

Attention : ces tailles (sauf pour char) sont dépendantes du système d'exploitation, du compilateur ou du processeur utilisé.

L'opérateur **sizeof**(un type) retourne le nombre d'octets utilisés pour stocker une valeur d'un type donné.

Les entiers non signés

Type	Taille en octets	Valeurs possibles	Format
Type	Taille	Valeurs possibles	Format
unsigned int	2 ou 4	de 0 à 65535 (/2 octets) de 0 à 4294967295 (/4 octets)	%u
unsigned short	2	de 0 à 65535	%u
unsigned long	4	de 0 à 4294967295	%lu

manipuler des entiers **non signés**, alors on ajoutera le mot « **unsigned** »

Booléen

- Le type **booléen** n'existe pas en C.
- Par contre
 - la valeur **0** (au sens binaire de la représentation) de n'importe quel type signifie **faux**.
 - Toute **autre valeur** est **vraie**.

Les réels

Type	Description	Taille en octets	Format
float	Réel	4	%f
double	Réel double précision	8	%lf

Les réels sont également appelés les flottants.

Les caractères

Type	Description	Taille en octets	Valeurs possibles	Format
char	Caractère (considéré comme entier signé)	1	de -128 à 127	%c
unsigned char	caractère (considéré comme entier non signé)	1	De 0 à 255	%c

- Caractères considérés comme des *entiers* codés en mémoire sur 1 octet.
- Chaque caractère est associé nombre entier = son code ASCII
- Plus que caractère imprimable : retours chariot, tabulations, sauts de page

Les caractères non imprimables

- Représentation conventionnelle utilisant le caractère `\` nommé **antislash**.
- Exemples de caractères non imprimables
 - `\n` retour chariot avec saut de ligne
 - `\r` retour chariot sans saut de ligne
 - `\t` tabulation horizontale
 - `\v` tabulation verticale

DÉCLARATION DE VARIABLES ET DE CONSTANTES

Déclaration d'une variable

- Avant première utilisation
- **Syntaxe**
 - *type identificateur;*
 - *type identificateur = valeur*
 - *type identificateur₁, identificateur₂, ...*
- **Exemple**
 - `int a ;`
 - `int b=10 ;`
 - `int x, var = 2, z ;`

Déclaration d'une constante

- **Syntaxe**

- *const type identificateur = valeur;*
- *#define identificateur valeur*

- **Exemple**

- *const int MAX=32767;*
- *#define MAX 32767*

- Dans le second cas pas de réservation d'emplacement mémoire

Contraintes sur la nomenclature

- Identificateur = suite de caractères contenant
 - des lettres (minuscules ou majuscules non accentuées)
 - des chiffres,
 - le caractère « souligné » (_).
- Premier caractère d'un identificateur ne peut pas être un chiffre.
- différence entre majuscules et minuscules.

Mots-clefs du C ANSI

Mots non utilisables comme identificateurs

auto	const	double	float	int	short	struct	unsigned
break	continue	else	for	long	signed	switch	void
case	default	enum	goto	register	sizeof	typedef	volatile
char	do	extern	if	return	static	union	while

OPÉRATEURS DU LANGAGE

Opérateurs arithmétiques

Opérateur	Signification
+	addition
-	soustraction
*	multiplication
/	division
%	modulo

Opérateurs relationnels

Opérateur	Signification
<	inférieur
<=	inférieur ou égal
>	supérieur
>=	supérieur ou égal
==	égal
!=	différent

Opérateurs logiques

Opérateur	Signification
&&	Et logique
	Ou logique
!	Négation logique

Opérateurs logiques

- **Exemples :**
- **$(a > 10) \&\&(a \leq 20) = \text{vrai}$**
 - si $a > 10$ et si $a \leq 20$
- **$(a > 10) || (b \leq 20) = \text{vrai}$**
 - si $a > 10$ ou si $b \leq 20$
- **$!(a > 10) = \text{vrai}$**
 - si $a > 10$ est faux , c-a-d si $a \leq 10$

INSTRUCTIONS ÉLÉMENTAIRES EN C

Affectation

- permet de donner la valeur de l'expression de droite à la variable de gauche.
- C utilise le **symbole =** pour l'affectation.
- **Exemples :**
 - **x = 5;** //La variable x prend pour valeur 5
 - **i = i + 1;** //La variable i prend pour valeur son ancienne valeur augmentée de 1
 - **C = 'a';** //La variable C prend pour valeur le caractère 'a'.

Incrémentation

- **Préincrémentation** : ++ (avant la variable)
 - Exemple : **++a;**
- **Postincrémentation** : ++ (après la variable)
 - Exemple : **a++;**
- **Prédécrémentation** : -- (avant la variable)
 - Exemple : **--a;**
- **Postdécrémentation** : -- (après la variable)
 - Exemple : **a--;**

Affichage

- Fonction **printf** afficher des infos à l'écran
- **Syntaxe**
 - **printf** (" chaîne de caractères " , var₁, var₂, ...) ;
- Dans la bibliothèque **stdio.h**

Affichage (suite)

- **Format d'affichage**
 - **%d** ou **%ld** ou **%u** ou **%lu** pour les entiers (int, short, long, unsigned)
 - **%f** ou **%lf** pour les réels (float, double)
 - **%s** pour les chaînes de caractères
 - **%c** pour les caractères

Affichage : exemple 1

- **printf**("La valeur des variables **X** et **Y** sont : %d et %d", X, Y) ;
- La chaîne peut contenir caractères spéciaux :
 - \n pour saut de ligne
 - \t pour les tabulations
- **printf** ("Bonjour\nCa va?") ; //affiche????

Bonjour
Ca va?

Affichage : exemple 2

- Pour $y=10$ et $x=5$
- `printf ("%d", y - x);` affiche ???

5

- `printf ("la valeur de x est %d et celle de y e st %d", x, y);` //Affiche ???

La valeur de x est 5 et celle de y est 10

Affichage : exemple 3

- `int i =1;`
- `float x = 2 . 0 ;`
- `printf (" Bonjour \ n ") ;`
- `printf (" i = %d \ n " , i) ;`
- `printf (" i = %d , x = % f \ n " , i , x) ; //Affiche ???`

```
Bonjour  
i = 1  
i = 1, x=2.0
```

Affichage (fin)

- **printf** offre la possibilité de définir le nombre d'emplacements à utiliser pour l'affichage d'une valeur
- **printf**("La valeur de X est : %4.3f ", 3.5268) ;

La valeur de X est :xxx3.527

- chaque **x** représentant un **espace**

Lecture

- **scanf** permet au programme de lire des informations saisies au clavier (**stdio.h**)
- **Syntaxe :**
scanf (" chaîne de formatage " , **&variable₁** , **&variable₂**, ...) ;
- Caractère **&** permet d'accéder à l'adresse d'une variable.
- A omettre dans le cas de la lecture d'une chaîne de caractères

Format de lecture

- **%d** ou **%ld** ou **%u** ou **%lu** pour les entiers (int, short, long, unsigned)
- **%f** ou **%lf** pour les réels (float, double)
- **%s** pour les chaînes de caractères
- **%c** pour les caractères

Format de lecture

- **Exemple :**
 - `scanf ("%d%d", &a , &b) ;`
 - Taper 2 puis 3, la variable a prend la valeur 2 et b prend la valeur 3.

Lecture / Affichage de caractère

- **getchar** et **putchar** de **stdio.h** permettent au programme de *lire* au clavier et *d'afficher* à l'écran
- **getchar** retourne un **entier** le code ASCII
 - Exemple : *char c = getchar();*
- **putchar**
 - Exemple : *putchar(c);*
- **fflush()**

Lecture / Affichage de caractère (bis)

- Certains compilateurs proposent la bibliothèque **conio.h** avec les fonctions **getch** et **putch**.
 - `char c = getch() ;`
 - `putch(c) ;`
- **getch** lit directement le caractère sans utiliser buffer d'entrée.
- Ces deux fonctions ne font pas partie du langage C standard.

Overview

1. PRÉSENTATION DU COURS
2. INTRODUCTION GÉNÉRALE
3. NOTIONS D'ALGORITHMIQUE
4. INTRODUCTION AU LANGAGE C
- 5. STRUCTURES DE CONTRÔLE**
6. TYPES DE DONNÉES COMPOSÉS
7. POINTEURS

CHAPITRE 4

STRUCTURES DE CONTRÔLE

OVERVIEW

1. Structures conditionnelles
2. Structures de répétition
3. Instructions de rupture de séquence en C

Introduction

- Agir sur l'ordre ou la fréquence d'exécution des instructions
- Deux grands types de structures de contrôle
 - *structures conditionnelles*
 - *structures répétitives*, encore appelées boucles
 - Instructions de rupture de séquence

STRUCTURES CONDITIONNELLES

La structure de bloc.

Pseudo-code	Langage C
<pre>debut instruction(s) fin</pre>	<pre>{ instruction(s); }</pre>

La structure Si

- **Si** conditionne l'exécution d'un ensemble d'instructions à la valeur d'une condition (expression booléenne)
- Syntaxe

Si (condition) **Alors**

 traitement₁

Sinon

 traitement₂

FinSi

Remarques

- Les traitements après les mots **Alors** et **Sinon** peuvent être simple ou = à un ensemble d'instructions (bloc d'instructions)
- La **condition** est d'abord évaluée.
 - Si **vraie**, *traitement*₁ exécuté puis **FinSi**
 - Si **fausse**, *traitement*₂ exécuté puis **FinSi**

Exemple 1

...

Ecrire("entrez un nombre")

Lire(n)

Si ($n > 0$) **Alors** // cas où la condition $n > 0$ est **vraie**

Ecrire("valeur positive")

Sinon //cas où la condition $n > 0$ est **fausse**

Ecrire ("valeur négative ou nulle")

FinSi

...

Exemple 2

Programme Operation

Variable

nb1,nb2, res: entier

op : caractère

Début

Ecrire("Entrez deux nombres")

Lire(nb1, nb2)

Ecrire("entrez la première lettre de l'opération : somme ou produit")

Lire(op)

Si (op == 's') **Alors**

res \leftarrow nb1 + nb2

Ecrire("la somme est", res)

Sinon

res \leftarrow nb1 * nb2

Ecrire("le produit est", res)

FinSi

Fin

Remarque 1

- Sinon non obligatoire

Si (condition) **Alors**
traitement
FinSi

Remarque 2

- Elle peut contenir d'autres alternatives

Si (condition₁) **Alors**

 traitement₁

Sinon Si (condition₂) **Alors**

 traitement₂

...

Sinon Si (condition_N) **Alors**

 traitement_N

Sinon

 traitement

FinSi

Traduction en C (1)

Pseudo-code	Langage C
si (condition) alors instruction(s); finsi	if (condition) { instruction(s); }

Traduction en C (2)

Pseudo-code	Langage C
<pre>si condition alors instruction(s)_1 sinon instruction(s)_2 finsi</pre>	<pre>if (condition) { instruction(s)_1; } else { instruction(s)_2; }</pre>

Traduction en C (3)

Pseudo-code	Langage C
<pre>si condition_1 alors instruction(s)_1 Sinon si condition_2 alors instruction(s)_2 sinon instruction(s)_3 finsi</pre>	<pre>if (condition_1) { instruction(s)_1; } else if (condition_2) { instruction(s)_2; } else { instruction(s)_3; }</pre>

La structure d'alternative (imbriquée)

Pseudo-code	Langage C
<pre>si condition_1 alors instruction(s)_1 sinon si condition_2 alors instruction(s)_2 sinon instruction(s)_3 finsi finsi</pre>	<pre>if (condition_1) { instruction(s)_1; } else { if (condition_2) { instruction(s)_2; } else { instruction(s)_3; } }</pre>

Exemple 1

```
...  
printf("entrez un nombre");  
scanf("%d", &n);  
if (n > 0)  
    printf("valeur positive");  
else  
    printf("valeur négative ou nulle");  
...
```

Exemple 2

```
#include <stdio.h>
int main( )
{
    int nb1,nb2, res;
    char op;
    printf("Entrez deux nombres");
    scanf("%d%d", &nb1, &nb2);
    printf("entrez la première lettre de l'opération voulue: somme ou produit");
    fflush(stdin);
    op = getchar( );
    if (op == 's')
    {
        res = nb1 + nb2;
        printf("la somme est %d", res);
    }
    else if(op == 'p')
    {
        res = nb1 * nb2;
        printf("le produit est %d", res);
    }else
    {
        printf("la lettre que vous avez entrée n'est ni somme ni produit");
    }
    return 0;
}
```

Structure Selon

- **Selon** choisit le traitement en fonction de la valeur d'une variable ou d'une expression.
- remplace avantageusement une structure **Si**.
- Syntaxe

```
Selon (expression) Faire  
    valeur1 : traitement1  
    valeur2 : traitement2  
    ...  
    valeurN : traitementN  
    Sinon traitement  
FinSelon
```

Remarques

- **expression** est un type scalaire
 - entier, caractère, booléen ou énuméré
- **expression** est évaluée, puis sa valeur est successivement comparée à chacune des valeurs.
- Si correspondance, arrêt comparaisons et traitement associé exécuté.
- Si aucune correspondance le traitement associé au **Sinon**, s'il existe, est exécuté

Exemple

...

Ecrire("Donner le numéro du mois")

Lire(mois)

Selon (mois) **Faire**

1 : **Ecrire**("Janvier")

2 : **Ecrire**("Février")

3 : **Ecrire**("Mars")

4 : **Ecrire**("Avril")

...

11: **Ecrire**("Novembre")

12: **Ecrire**("Décembre")

Sinon Ecrire("Un numéro de mois doit être compris entre 1 et 12")

FinSelon

...

Exemple avec Si

...

Ecrire(Donner le numéro du mois)

Lire(mois)

Si (mois == 1) **Alors**

Ecrire("Janvier")

Sinon Si (mois == 2) **Alors**

Ecrire("Février »)

...

Sinon Si (mois == 12) **Alors**

Ecrire("Décembre")

Sinon

Ecrire("Un numéro de mois doit être compris entre 1 et 12")

FinSi

...

Traduction en C

Pseudo-code	Langage C
selon variable cas valeur_1 instruction(s)_1 cas valeur_2 instruction(s)_2 ... cas sinon instruction(s)_sinon fselon	switch (variable) { case valeur_1 instruction(s)_1; break; case valeur_2 instruction(s)_2; break; ... default : instruction(s)_sinon }

Exemple en C

...

printf("Donner le numéro du mois")

scanf("%d", &mois)

switch (mois)

{

case 1 : printf("Janvier"); **break**;

case 2 : printf("Février"); **break**;

...

case 12 : printf("Décembre"); **break**;

default : printf("Un numéro de mois doit être compris entre 1 et 12"); **break**;

}

...

STRUCTURES RÉPÉTITIVES

Introduction

- Appelées **boucles** permettent de répéter un traitement (c'est à dire une instruction simple ou composée) autant de fois qu'il est nécessaire
 - soit un nombre déterminé de fois
 - soit tant qu'une condition est vraie

Boucle Tant que

- **Tant que** répète un traitement tant qu'une condition est vraie.
- Si, dès le début, la condition est **fausse**, alors le **traitement** ne sera pas exécuté.
- **Syntaxe**

Tant que (condition) **Faire**
 traitement
FinTQ

Exemple : calcul du cube de nombres non nuls

Programme cube

Variable x : Entier

Début

Ecrire("Ce programme calcul le cube des nombres que vous entrez.
Pour arrêter tapez 0.")

Ecrire("Entrez un nombre")

Lire(x)

Tant que (x ≠ 0) **Faire**

Ecrire("le cube de " , x , " est " , x*x*x)

Ecrire("Entrez un nombre ou 0 pour arrêter")

Lire(x)

FinTQ

Ecrire("Fin")

Fin

Traduction en C

Pseudo-code	Langage C
tq condition faire instruction(s) fintq	while (condition) { instruction(s); }

Exemple : calcul du cube de nombres non nuls

```
void cube ()
{
    int x;
    printf("Ce programme calcul le cube des nombres que vous entrez.  
Pour arrêter tapez 0.\n");
    printf("Entrez un nombre");
    scanf("%d", &x);
    while (x != 0) {
        printf("le cube de %d est %d" , x , x*x*x);
        printf("Entrez un nombre ou 0 pour arrêter");
        scanf("%d", &x);
    }
    printf("Fin");
}
```

Boucle Pour

- La boucle **Pour** permet de répéter une instruction un nombre connu de fois. Elle a le formalisme suivant:
- **Syntaxe**

Pour *compteur* \leftarrow *valeur_initiale* à *valeur_finale* **par pas de** *increment*
Faire
 traitement
FinPour

Exemple

- Affichage des nombres pairs de 0 à 20

...

Pour $x \leftarrow 0$ à 20 par pas de 2 Faire

Ecrire(x)

FinPour

...

- Affichage des nombres pairs de 20 à 0

...

Pour $x \leftarrow 20$ à 0 par pas de -2 Faire

Ecrire(x)

FinPour

...

Comparaison avec Tant que

- Avec initialisation

...

$x \leftarrow 1$

Tant que ($x \leq 20$) **Faire**

Ecrire(x)

$x \leftarrow x+1$ // incrémentation explicite

FinTQ

...

Traduction en C (pas de 1)

Pseudo-code	Langage C
<p>pour i de d à f faire instruction(s) finpour</p>	<pre>int i ; for (i= d ; i<= f ; i ++) { instruction(s); } OU BIEN for (int i = d ; i<= f ; i ++) { instruction(s); }</pre>

La boucle Pour (pas de -1)

Pseudo-code	Langage C
<p>pour i de d à f faire instruction(s) finpour</p>	<pre>int i ; for (i= d ; i<= f ; i --) { instruction(s); } OU BIEN for (int i = d ; i<= f ; i --) { instruction(s); }</pre>

La boucle Pour (pas de +p)

Pseudo-code	Langage C
<p>pour i de d à f pas p faire instruction(s) finpour</p>	<pre>int i ; for (i= d ; i<= f ; i +=p) { instruction(s); } OU BIEN for (int i = d ; i<= f ; i +=p) { instruction(s); }</pre>

La boucle Pour (pas de -p)

Pseudo-code	Langage C
<p>pour i de d à f pas p faire instruction(s) finpour</p>	<pre>int i; for (i= d ; i<= f ; i -=p) { instruction(s); } OU BIEN for (int i = d ; i<= f ; i -=p) { instruction(s); }</pre>

Exemple

```
for(x = 0; x <= 20; x = x+1)
{
    printf("%d", x);
}
```

- **Nb** : A la sortie de la boucle, x vaut 21 mais ne sera pas affichée.

Boucle Faire ... Tant que

- Répète une instruction tant qu'une condition est vraie.
- Sa syntaxe est la suivante :

Faire

traitement

Tant que (*condition*)

Exemple

Programme Aire

Variable

rayon : réel

réponse : caractère

Début

Ecrire("Calcul de l'aire d'un cercle")

Faire

Ecrire("Entrez le rayon d'un cercle en cm")

Lire(rayon)

Ecrire("L'aire de ce cercle est ", rayon*rayon *3.14, "cm2")

Ecrire("Voulez-vous calculer l'aire d'un autre cercle? (o/n)")

Lire(réponse)

Tant que (réponse == 'o')

Ecrire("Au revoir!")

Fin

Traduction en C

Pseudo-code	Langage C
faire instruction(s) Tant que condition	do { instruction(s) ; } while (condition) ;

INSTRUCTIONS DE RUPTURE DE SÉQUENCE EN LANGAGE C

L'instruction break

- Utilisable dans n'importe quelle boucle.
- Interruption déroulement de la boucle
- passage à la première instruction qui suit la boucle.
- En cas de boucles imbriquées, **break** fait sortir de la boucle la plus interne.

Exemple

```
int main()
{
    int i;
    for (i = 0; i < 5; i++)
    {
        printf("i = %d\n",i);
        if (i == 3)
            break;
    }
    printf("valeur de i a la sortie de la boucle = %d\n",i);
    return 0;
}
```


Résultat de l'exemple

- $i = 0$
- $i = 1$
- $i = 2$
- $i = 3$
- valeur de i à la sortie de la boucle = 3

L'instruction continue

- **continue** permet de passer directement à l'étape suivante d'une boucle
- sans exécuter les autres instructions de la boucle qui la suivent

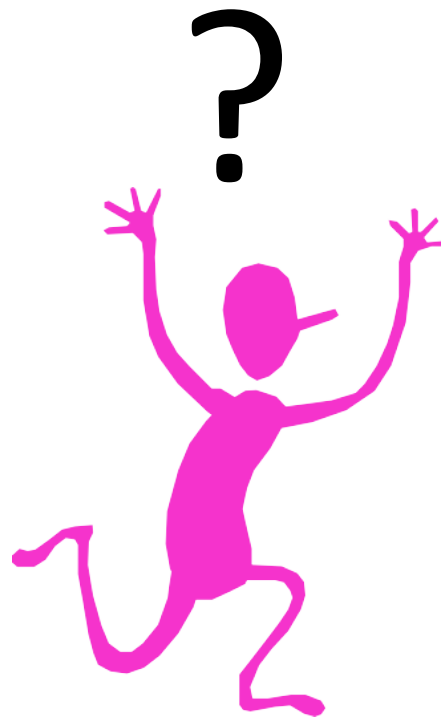
Exemple

```
int main()
{
    int i;
    for (i = 0; i < 5; i++)
    {
        if (i == 3)
            continue;
        printf("i = %d\n",i);
    }
    printf("valeur de i a la sortie de la boucle = %d\n",i);
    return 0;
}
```

Résultat de l'exemple

- $i = 0$
- $i = 1$
- $i = 2$
- $i = 4$
- valeur de i a la sortie de la boucle = 5

END



Overview

1. PRÉSENTATION DU COURS
2. INTRODUCTION GÉNÉRALE
3. NOTIONS D'ALGORITHMIQUE
4. LANGAGE C
5. STRUCTURES DE CONTRÔLE
- 6. TYPES DE DONNÉES COMPOSÉS**
7. POINTEURS

Chapitre 5

TYPES DE DONNEES COMPOSÉS

Introduction

- Créer avec les types de base
 - caractère
 - entier
 - Réel
- De nouveaux types, appelés *types composés*
- Représenter ensembles de données organisés

Overview

1. Énumérations
2. Tableaux à une dimension
3. Tableaux à plusieurs dimensions
4. Chaînes de caractères
5. Structures

ÉNUMÉRATIONS

Pourquoi les énumérations?

- L'utilisation de valeurs alphanumériques (lettres ou chiffres) est courante pour coder de l'information, par exemple :
 - 'F' pour féminin, 'M' pour masculin,
 - **note[0]** pour la note d'examen d'intra
 - **note[1]** pour la note d'examen final
 - **note[2]** pour la note du premier travail pratique
 - ...
 - **note[5]** pour la note globale des travaux pratiques
 - **note[6]** pour la note globale du cours

Cependant ...

- Le **C** permet de donner plus de clarté au codage de l'information en utilisant le type énumération :
 - 1. `char poste = Programmeur ;`
est plus clair que `Poste = 'P' ;` (est-il polygame?)
 - 2. ***note[tps]*** est plus significative que ***note[5]***.

Introduction

- **Définition** : Une énumération est un type permettant de définir un ensemble de constantes, parmi lesquelles les variables de ce type prendront leur valeur.
- Pour déclarer une variable de type énuméré, il faut d'abord créer le type.

Définition en algorithmique

- type

$nom_type = \{constante_1, constante_2, \dots, constante_N\}$

- nom_type = identificateur du nouveau type
- $constante_1, constante_2, \dots, constante_N$ = liste identificateurs donnant l'ensemble des valeurs de ce type.

Exemple

- **type**

// définition du type couleur

couleur = {*bleu, blanc, rouge, vert, jaune, noir*}

// définition du type jour

jour = {*lundi, mardi, mercredi, jeudi, vendredi, samedi, dimanche*}

Déclaration de variables

- Définition type énuméré → utiliser comme un type normal
- Déclarer une ou plusieurs variables de ce type
- **Exemple :**

...

variable

c : couleur // déclaration de la variable c de type couleur

Début

...

c ← bleu // utilisation de la variable c

Fin

En langage C

- Définition type énuméré
 - mot-clé **enum**
- Syntaxe
enum nom_type {*constante₁*, *constante₂*, ..., *constante_N*};
- Exemples:
 - **enum couleur** {*bleu, blanc, rouge, vert, jaune, noir*} ;
 - **enum jour** {*Lundi, Mardi, Mercredi, Jeudi, Vendredi, Samedi, Dimanche*} ;

Remarques

- Chaque constante est associée à un entier
 - son numéro d'ordre (partant de 0 pour le 1^{er})
 - on peut aussi spécifier la valeur associée.
- **Exemple:**
 - `enum jour {Lundi=1, Mardi, Mercredi=5, Jeudi, Vendredi, Samedi, Dimanche};`
 - Mardi prend la valeur 2, Jeudi la valeur 6, ...
- **Remarque** : Les valeurs associées aux constantes ne sont pas modifiables dans le programme.

Déclaration de variables

- Après avoir défini un type énuméré, on peut l'utiliser pour déclarer une ou plusieurs variables de ce type.
- **Exemples:**
 - `enum jour j1;`
 - `enum jour j2 = Mardi;`

Exemple

```
#include <stdio.h>
#include <conio.h>
enum Jour {lundi=1,mardi,mercredi,jeudi,vendredi,samedi,dimanche};
int main()
{
    enum Jour un_jour;
    printf("Donnez un numéro de jour entre 1 et 7") ;
    scanf("%d", &un_jour) ;
    switch(un_jour )
    {
        case lundi :
            printf("C'est Lundi"); break ;
        case mardi :
            printf("C'est Mardi"); break ;
        case mercredi :
            printf("C'est Mercredi"); break ;
        default :
            printf("Ce n'est ni Lundi ni Mardi ni Mercredi");
    }
    getch();
    return 0;
}
```



Exemple

```
#include <stdio.h>
#include <conio.h>
enum mois {jan=1,fev,mars,avril,mai,juin,juil, aout, sept, oct, nov, dec};
int main()
{
    enum mois un_mois;
    printf("Donnez un numéro de mois entre 1 et 12");
    scanf("%d", &un_mois);
    switch(un_mois)
    {
        case jan :
            printf("C'est Janvier"); break ;
        case fev :
            printf("C'est Février"); break ;
        case mars :
            printf("C'est Mars"); break ;
        case avril :
            printf("C'est Avril"); break ;
        case mai :
            printf("C'est Mai"); break ;
        case juin :
            printf("C'est Juin"); break ;
        case juil:
            printf("C'est Juillet"); break ;
        case aout :
            printf("C'est Aout"); break ;
        case sept :
            printf("C'est Septembre"); break ;
        case oct :
            printf("C'est Octobre"); break ;
        case nov :
            printf("C'est Novembre"); break ;
        case dec :
            printf("C'est Décembre"); break ;
        default :
            printf("Ce n'est pas un numéro de mois valide");
    }
    getch();
    return 0;
}
```

TABLEAUX

Problématique

- Supposons que nous ayons 2 notes
 - Déclarer 2 variables; OK
- Supposons que nous ayons 20 notes
 - Déclarer 20 variables; ???
- Supposons que nous ayons 200 notes
 - Déclarer 200 variables; ????

Exemple 1

- Supposons que nous ayons 12 notes
- Déclarer 12 variables
 - Succession de 12 instructions **lire**
 - $\text{moy} \leftarrow (n_1 + n_2 + \dots + n_{12}) / 12$
- Avec des centaines ou milliers de valeurs
 - suicide direct
- rassembler toutes ces variables en une seule
 - tableau

Définitions

- Une **variable indicée**
- Un **vecteur**
 - Ensemble de valeurs portant le même nom de variable et repérées par un nombre
 - Le nombre qui, au sein d'un tableau, sert à repérer chaque valeur s'appelle **indice**.
 - Désigner un élément du tableau, on fait figurer le nom du tableau, suivi de l'indice de l'élément

TABLEAUX À UNE DIMENSION

Définition

- Exemple

10000	8500	12300	13000	6500	9800
1 ^{ère} case	2 ^{ème} case	3 ^{ème} case	4 ^{ème} case	5 ^{ème} case	6 ^{ème} case

- Nom : salaire
- Taille (nombre d'éléments) : 6

- Remarques

- « case contenant une valeur » doit faire penser à celle de variable
- éléments d'un tableau correspondent à des emplacements contiguës en mémoire

Déclaration

Variable

nom_tableau : tableau [*N*] de *type*

Où

- *nom_tableau* est l'identificateur du tableau
- *N* est la taille du tableau et doit être une constante entière et positive.
- *type* est le type des éléments du tableau

Exemple

- **variable**
 - salaire : tableau[6] de réel
 - nom_clients: tableau [20] de caractere
 - notes : tableau[8] d'entier
- Un élément particulier du tableau est désigné en précisant son **indice**
- 1^{er} élément du tableau → 0 et est désigné par *nom_tableau* [0]
- 2^e élément du tableau → 1 et est désigné par *nom_tableau* [1]
- ...
- Le dernier élément du tableau → N-1 et est désigné *nom_tableau* [N-1]

Exemple (suite)

10000	8500	12300	13000	6500	9800
0	1	2	3	4	5

- Pour désigner un élément, l'indice peut être écrit sous forme de :
 - Nombre en clair, exemple: **salaire [4]**
 - Variable, exemple: **salaire [i]**, avec i de type entier
 - Expression entière exemple : **salaire [k+1]**, avec k de type entier

Remarques

- V (forme) valeur de l'indice
 - entière
 - comprise entre les valeurs minimale et maximale
 - Exemple, avec le tableau salaire, interdit d'écrire
 - `salaire[10]` ou `salaire[15]`.
 - Expressions font référence à des éléments qui n'existent pas
- Mélanger indice et valeur : 3^e maison de la rue n'a pas forcément 3 habitants, et la 20^e 20
- Aucun lien en `i` et `salaire[i]`

Exemple

Algorithme gestab

variable

Note : **Tableau** [12] de réel

Moy, Som : réel

Début

Pour $i \leftarrow 0$ à 11 faire

Ecrire ("Entrez la note n°", i)

Lire Note(i)

Finpour

Som $\leftarrow 0$

Pour $i \leftarrow 0$ à 11 faire

Som \leftarrow Som + Note(i)

Finpour

Moy \leftarrow Som / 12

Fin

Algorithme gestab

variable

Note : **Tableau** [12]
de réel

Moy, Som : réel

Début

Som $\leftarrow 0$

Pour $i \leftarrow 0$ à 11 faire

Ecrire ("Entrez la note n°", i)

Lire Note(i)

Som \leftarrow Som + Note(i)

Finpour

Moy \leftarrow Som / 12

Fin

En Langage C

- Syntaxe
 - ***type*** *nom_tableau* [*N*] ;
 - *nom_tableau* est l'identificateur du tableau
 - *N* = taille tableau est constante entière positive
- Exemple
 - `int tab[10] ; //déclare une variable tableau de 10 entiers appelée tab`
 - Réservation emplacement mémoire stocker 10 entiers
 - `tab[0]` désigne le premier élément du tableau `tab`
 - `tab[9]` désigne le dernier élément du tableau `tab`

Exemple

```
int main() {  
    float Note [12];  
    float Moy, Som;  
  
    for (int i=0; i<12; i++) {  
        printf("Entrez la note n°%d", i);  
        scanf( "%f", &Note[i]);  
    }  
    Som = 0;  
    for (int i=0; i<12; i++ )  
        Som = Som + Note[i];  
  
    Moy = Som / 12;  
    printf("la moyenne est = %f", Moy);  
    return 0;  
}
```

Initialisation

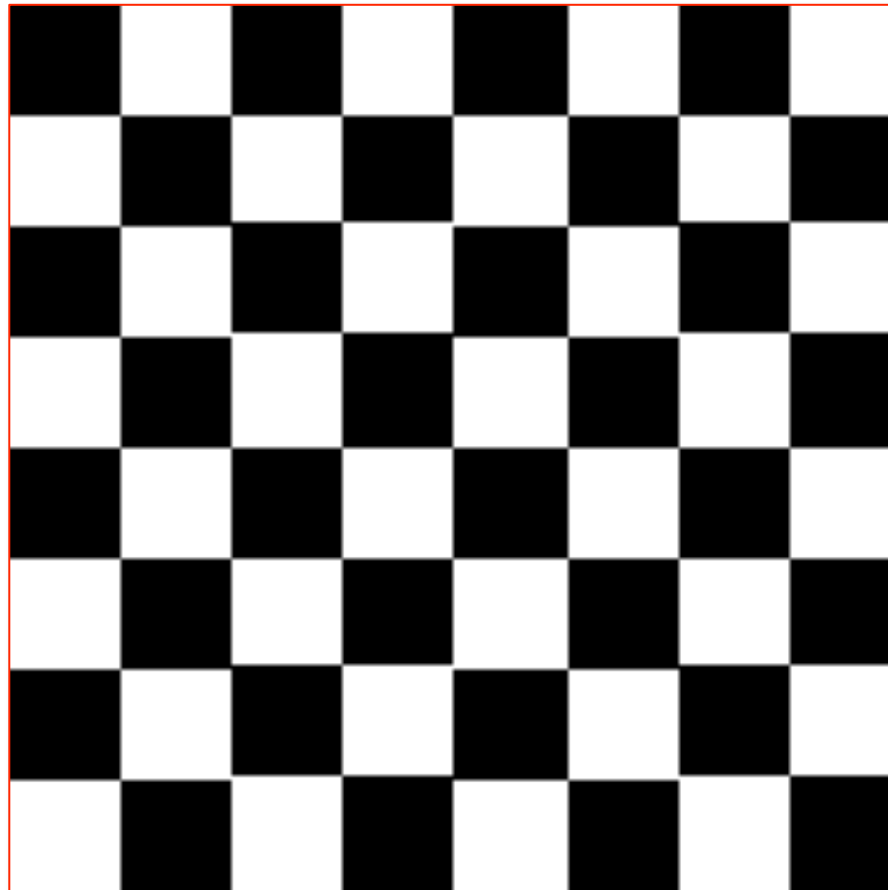
- `int T[] = {4,5,8,12,-3};`
 - Déclaration du tableau T de taille fixée à 5.
 - initialisation des éléments du tableau
 - Indication du nombre d'éléments à l'intérieur des crochets non obligatoire

Exemple

```
int main() {  
  
    float Note [] = {12, 3, 14, 9};  
    float Moy, Som;  
  
    Som = 0;  
    for (int i=0; i<4; i++ )  
        Som = Som + Note[i];  
  
    Moy = Som / 4;  
    printf("la moyenne est = %f", Moy);  
  
    return 0;  
  
}
```

TABLEAUX À PLUSIEURS DIMENSIONS





Introduction

- Plus adapté à certains problèmes !
- Jeu de dames sur un damier de 64 cases
- Modélisé par un tableau de 64 valeurs
- Bien sûr, on peut programmer tout un jeu
- Mais on sent que ce n'est pas simple et qu'il serait plus facile de modéliser un damier par un... damier !

Problématique

- **Pourquoi?** Modélisation de damier en une dimension
 - 1 à 8 : première ligne
 - 9 à 16 : deuxième ligne
 - ...
 - 57 à 64 : dernière ligne
- Dans case on met **1** si présence d'un pion et **0** sinon
 - $\text{case}(i) \rightarrow \text{case}(i+7), \text{case}(i+9), \text{case}(i-7), \text{case}(i-9)$.
- On peut résoudre le problème plus simplement
→ modéliser un damier par un damier

Le damier

- Tableaux à 2 dimensions
 - Valeurs repérées par deux coordonnées
- Déclaration
 - **Damier : tableau[8][8] d'entier**
 - Réservation de **64 entiers (8x8)**
 - **Damier (i, j) → Damier (i-1, j-1), Damier (i-1, j+1), Damier (i+1, j-1) et Damier (i+1, j+1)**

Remarques

- Aucune différence qualitative entre un tableau à **1** dimension et un autre à **2**
- Pas de lignes ni de colonnes
- Tableaux à **n** dimensions
 - **mingming**(3, 5, 4, 4) $\rightarrow 3 \times 5 \times 4 \times 4 = 240$ valeurs. Chaque valeur *y* est repérée par **4** coordonnées.
 - Rarement **n > 3**
 - Matheux allez y car vous n'êtes pas comme nous
 - Habitues à l'espace à plusieurs dimensions

Création

- Déclaration
 - Plusieurs indices pour désigner un élément
 - Paire de crochets et taille pour chaque dimension
- Exemple
variable
nom_tableau: tableau[N1][N2] de type

Exemple

2	1
-4.5	23
0	9

- Matrice = tableau à 2 dimensions
- `tab[0][0]` → élém à la ligne 0 et à la colonne 0
 - Vaut 2
- `tab[2][1]` → élém à la ligne 2 et à la colonne 1
 - Vaut 9

Remarques

- Utiliser **deux boucles Pour imbriquées** pour **parcourir tous les éléments d'un tableau à deux dimensions** : la 1^{ère} boucle pour une dimension et la 2^e pour l'autre dimension
- Possible définir un type tableau et l'utiliser
- **Exemple :**
 type
 tab : **tableau**[3] [6] de Réel
 variable
 T₁, T₂ : tab // T₁ et T₂ sont des variables de type tab

Exemple : Résultat?

- Variables
 - X : Tableau [2,3] en Entier;
 - i, j, val en Entier;
 - Début
 - $val \leftarrow 1$;
 - Pour $i \leftarrow 0$ à 1
 - Pour $j \leftarrow 0$ à 2
 - $X(i,j) \leftarrow val$;
 - $val \leftarrow val + 1$;
 - Finpour
 - Finpour
 - Pour $i \leftarrow 0$ à 1
 - Pour $j \leftarrow 0$ à 2
 - Ecrire $X(i,j)$;
 - Finpour
 - Finpour
 - Fin
- Variables
 - X : Tableau [2,3] en Entier;
 - i, j, val en Entier;
 - Début
 - $val \leftarrow 1$;
 - Pour $i \leftarrow 0$ à 1
 - Pour $j \leftarrow 0$ à 2
 - $X(i,j) \leftarrow val$;
 - $val \leftarrow val + 1$;
 - Finpour
 - Finpour
 - Pour $j \leftarrow 0$ à 2
 - Pour $i \leftarrow 0$ à 1
 - Ecrire $X(i,j)$;
 - Finpour
 - Finpour
 - Fin

Exemple : Résultat?

- Variables
 - X : Tableau [2,3] en Entier;
 - i, j, val en Entier;
- Début
 - $val \leftarrow 1$;
 - Pour $i \leftarrow 0$ à 1
 - Pour $j \leftarrow 0$ à 2
 - $X(i,j) \leftarrow val$;
 - $val \leftarrow val + 1$;
 - Finpour
 - Finpour
 - Pour $j \leftarrow 0$ à 2
 - Pour $i \leftarrow 0$ à 1
 - Ecrire $X(i,j)$;
 - Finpour
 - Finpour
- Fin

En Langage C

- Ajouter paire de crochets et une taille pour chaque dimension.

- Syntaxe

type nom_tableau[N1][N2];

- **Exemple :**

- **double m[10][20];** */*m est une matrice de réels*/*
- **m[0][0]** désigne l'élément à la ligne 0, colonne 0
- **m[9][19];** désigne l'élément à ligne 9, colonne 19

Exemples : initialiser et compter le nombre de zéros dans la matrice

```
Int main(){
    Float mat[4][5];
    Int i=0, j, zero;
    For(; i<4; i++){
        For (j=0; j<5; j++)
            Mat[i][j] = j-i;
    }
    Zero = 0;
    For (i=0; i<4; i++){
        For (j=0; j<5; j++)
            If (mat[i][j] == 0)
                zero+=1;
    }
    Printf("le nombre d'éléments nuls est %d", zero);
    Return 0
}
```

Produit matricielle

- $\forall i,j :$
$$c_{ij} = \sum_{k=1}^n a_{ik} * b_{kj}$$

- Exemple

$$\begin{pmatrix} 1 & 0 \\ -1 & 3 \end{pmatrix} \times \begin{pmatrix} 3 & 1 \\ 2 & 1 \end{pmatrix} = \begin{pmatrix} (1 \times 3 + 0 \times 2) & (1 \times 1 + 0 \times 1) \\ (-1 \times 3 + 3 \times 2) & (-1 \times 1 + 3 \times 1) \end{pmatrix} = \begin{pmatrix} 3 & 1 \\ 3 & 2 \end{pmatrix}$$

CHAÎNES DE CARACTÈRES

Introduction

- Chaîne de caractère gérée en langage C comme un tableau contenant des caractères
- Particularité : la dernière case du tableau utilisée pour la chaîne contient le **caractère spécial \0** appelé **caractère nul**
- Caractère représente la fin de la chaîne

Déclaration

- Tableau de caractères
- **Syntaxe**
 - `char nom_chaine [N] ;`
 - (dans ce cas limitée à N-1 caractères (+ '\0'))
- **Exemple**
 - Déclarer chaîne de 5 caractères
 - utiliser un tableau de taille 6
 - `char ch[6] ;`
 - **ch** est dans ce cas limitée à 5 caractères (+ \0)

Exemple

- Comme les tableaux numériques, on peut initialiser un tableau de caractères (ou chaîne de caractères) lors de sa déclaration
- **Exemple**
 - `char ch[] = "bonjour";`
 - le compilateur réserve un tableau de 8 octets
 - 7 octets pour "bonjour"
 - 1 octet pour le caractère de fin de chaîne `'\0'`.

Exemple (suite)

'b'	'o'	'n'	'j'	'o'	'u'	'r'	\0
-----	-----	-----	-----	-----	-----	-----	----

- On peut aussi écrire:

```
char ch[] = {'b','o','n','j','o','u','r'};
```


Fonctions des chaînes (1)

- La bibliothèque standard **string.h**
 - **strlen** → longueur (nombre caractères chaîne)
Cette fonction ne prend pas en compte le caractère `\0`
 - **strlen("bonjour");** // retourne 7

Fonctions des chaînes (1)

- La bibliothèque standard **string.h**
 - **strcpy** → affectation
 - **strcpy**(ch, "hello") ; // ch="hello";
 - **strncpy** → affectation de **lgmax** caractères de "hello" à **ch** (en complétant par **\0** éventuellement)
 - **strncpy**(ch, "hello everybody", lgmax) ; // ch="hello e"
si lgmax = 7

Fonctions des chaînes (2)

- `ch="hello "`
- **`strcat`** → concaténation de deux chaînes
 - **`strcat`** (`ch`, `"world"`); // `ch = "hello world"`
- **`strncat`** → concaténation de deux chaînes
 - `ch="hello "`
 - **`strncat`** (`ch`, `"world"`, `lgmax`); // `ch = "hello wo"` si `lgmax = 2`

Fonctions des chaînes (2)

- **strcmp** → comparaison de deux chaînes
 - **n = strcmp(ch1, ch2) ;**
 - **n** vaudra :
 - un **nombre négatif** si **ch1 < ch2** au sens syntaxique
 - **0** si **ch1** et **ch2** sont identiques
 - un **nombre positif** si **ch1 > ch2**
- **strncmp** → comme strcmp mais se limite à lgmax caractères
 - **n = strncmp(ch1, ch2, lgmax) ;**

STRUCTURES

Introduction

- Une structure désigne sous un seul nom un ensemble d'éléments pouvant être de types différents.
- Elle est un agrégat de données de types plus simples.
- Elle permette de construire des types complexes à partir des types de base ou d'autres types complexes

Construction

- Composée d'éléments appelés champ ou membre désignés par des identificateurs
- Types donnés dans la déclaration de la structure
- Types peuvent être n'importe quel autre type, même une structure.
- Les variables de type structure sont aussi appelées structures

En algorithmique

Déclaration

Type

nom_type = **Structure**

champ₁ : type₁

...

champ_N : type_N

FinStructure

- où
 - *nom_type* est l'identificateur du nouveau type
 - *type₁*, ..., *type_N* sont les types respectifs des champs *champ₁*, ..., *champ_N*

Exemples

- Les types **date** et **complexe** :

- **type**

- date = **Structure**

- jour : **entier**

- mois : **chaîne**

- année : **entier**

- FinStructure**

- complexe = **Structure**

- re : **réel**

- im : **réel**

- FinStructure**

Déclaration variable

- Comme tous les autres types
- Déclarer des variables
- Exemples:

Variable

d : date // d est une variable de type date

z : complexe // z est une variable de type complexe

Opérations

- Accès aux champs grâce à l'opérateur *point* **'.'**
- Exemple: **$x.champ_1 = champ_1$** d'une variable structure x
- Opérations valides sur le champ = opérations valides sur le type du champ
- Opérateur d'affectation valide (# tableau)
 - Copier tous les champs de la structure

Exemple

Programme date

Type

date = **Structure**

jour : **entier**

mois : **chaîne**

année : **entier**

FinStructure

Variable

d1, d2 : date

année : **entier**;

Début

// initialiser la date d1

d1.jour ← 23

d1.mois ← "Novembre"

d1.année ← 2000

// initialiser la date d2 à partir de d1

d2 ← d1

// afficher la date d2

Ecrire(d2.jour, '/', d2.mois, '/', d2.année)

// copier le champ année de d2 dans la variable année

année ← d2.année

// saisir le champ année de d2

Lire(d2.année)

Fin

Langage C

Création de structure

- Mot-clé **struct**

- Syntaxe

```
struct nom_type{  
    Type1 nom_champ1;  
    ...  
    TypeN nom_champN;  
};
```

- Exemple

```
struct complexe  
{  
    float réelle ;  
    float imaginaire ;  
};
```

Déclaration variable

- **Syntaxe** : `struct complexe c;`
 - Accès champs d'une variable structure avec `.`
 - Nom de la variable suivi de `.` + nom du champ
- **Exemple:**
 - *c.réelle = 0;*
 - *c.imaginaire = 1;*

Remarque 1

- Possible de déclarer variable structure sans créer au préalable le type structure
- Syntaxe

```
struct nomStructure  
{  
    Type1 champ1;  
    Type2 champ2;  
    ...  
} nomVariable;
```

Remarque 2

- Mêmes règles d'initialisation lors de la déclaration que pour les tableaux.

- **Exemple 1**

`struct complexe z = {2. , 2.};`

- **Exemple 2**

`struct complexe z1 = {2. , 2.};`

`struct complexe z2;`

`z2 = z1;`

COMPLÉMENTS

Opérateur **typedef**

- Alléger l'écriture des programmes
- Attribuer un nouvel identificateur à un type existant à l'aide du mot clé **typedef**
- Types de base | énumérations | structures
- Syntaxe
typedef type synonyme;

Exemple 1

- ***enum*** *couleur* {bleu, blanc, rouge, vert, jaune, noir} ;
- ***typedef*** *enum couleur* **couleur** ; // **couleur** devient le **synonyme** de **enum couleur**

```
int main()  
{  
    couleur c ; // c est une variable de type couleur  
    ...  
}
```

Exemple 2

```
struct complexe  
{  
    double reelle;  
    double imaginaire;  
};
```


```
typedef struct complexe complexe; //complexe est synonyme de  
struct complexe
```

```
int main()  
{  
    complexe z;  
    ...  
}
```

Type tableau

- Possibilité de définir un type tableau de la même manière qu'on déclare une variable tableau mais en écrivant d'abord le mot **typedef**.

- **Exemple :**

```
typedef  int tab[10]; // tab est un "type" tableau  
de 10 entiers
```

```
tab T; // T est une variable tableau de 10 entiers
```

Opérateur sizeof

- Argument
 - type ou
 - nom de variable
- Retour
 - taille (en octets) de l'espace mémoire nécessaire pour stocker une valeur de ce type.
 - **int i = sizeof(int); /* i vaut 4 */**

Fonctions Maths

- **SIN** double sin (double x)
- **COS** double cos (double x)
- **TAN** double tan (double x)
- **ASIN** double asin (double x)
- **ACOS** double acos (double x)
- **ATAN** double atan (double x)
- **ATAN2** double atan2 (double y, double x)
 - Fournit la valeur de $\arctan(y/x)$


Fonctions Maths

- **SINH double sinh (double x)**
 - Fournit la valeur de $\text{sh}(x)$
- **COSH double cosh (double x)**
 - Fournit la valeur de $\text{ch}(x)$
- **TANH double tanh (double x)**
 - Fournit la valeur de $\text{th}(x)$
- **EXP double exp (double x)**
- **LOG double log (double x)**
 - Fournit la valeur du logarithme népérien de x : $\text{Ln}(x)$ (ou $\text{Log}(x)$)
- **LOG10 double log10 (double x)**
 - Fournit la valeur du logarithme à base 10 de x : $\text{log}(x)$

Fonctions Maths

- **POW double pow (double x, double y)**
 - Fournit la valeur de xy
- **SQRT double sqrt (double x)**
- **CEIL double ceil (double x)**
 - Fournit (sous forme d'un double) le plus petit entier qui ne soit pas inférieur à x .
- **FLOOR double floor (double x)**
 - Fournit (sous forme d'un double) le plus grand entier qui ne soit pas supérieur à x .
- **FABS double fabs (double x)**
 - Fournit la valeur absolue de x .

Opérateur conditionnel

- `if (a>b)`
 - `max = a ;`
- `else`
 - `max = b ;`
-  `max = a>b ? a : b // affectation`
- `a>b ? i++ : i-- ; // sans affectation`

Exemple

- Calcul de la valeur absolue d'une expression
 - $3*a+1 > 0 ? 3*a+1 : -3*a-1$
- Cas où parenthèse obligatoire
 - $z = (x=y) ? a : b$
 - Affecter y à x ($x \leftarrow y$)
 - Si cette valeur # zéro ($z \leftarrow a$)
 - Sinon ($z \leftarrow b$)

Exemple (suite)

- $z = x = y ? a : b$
 - Est évaluée comme
- $z = x = (y ? a : b)$

END

