

# Introduction

A. Lambert/P. Courtieu

DSP - USAL 34

# Planning du cours

Date	heure	type	Intervenant	Programme
02/10/24	8:45	CM	A. Lambert	Introduction au langage C
02/10/24	13:30	TP	M. Boyer	Introduction au langage C
03/10/24	8:45	CM	A. Lambert	Fonctions, procédures et conditionnelle
03/10/24	13:30	TP	M. Boyer	Fonctions, procédures et conditionnelle (TP1)
04/10/24	8:45	TP	M. Boyer	Fonctions, procédures et conditionnelle (TP1)
04/10/24	13:30	TP	A. Lambert	Les boucles et les tableaux
10/10/24	8:45	TP	M. Boyer	Les boucles et les tableaux (TP1)
10/10/24	13:30	TP	M. Boyer	Les boucles et les tableaux (TP2)
11/10/24	8:45	CM	A. Lambert	Passage de paramètres et fonctions
11/10/24	13:30	TP	M. Boyer	Passage de paramètres et fonctions (TP1)
17/10/24	8:45	TP	M. Boyer	Passage de paramètres et fonctions (TP2)
17/10/24	13:30	TP	A. Lambert	Mini Projet
18/10/24	8:45	TP	M. Boyer	Mini Projet
18/10/24	14:00	CM	A. Lambert	Examen

Supports de cours, TD et TP sur <https://lecnam.net> ou <https://cedric.cnam.fr/~lamberta/enseignements/DSP/C>.

# Modalités d'évaluation

- Tous les TPs sont à rendre  
→ la moyenne compte pour 25% de la note finale.
  
- Un DS → la note compte pour 75% de la note finale.

# La chaîne de production de programmes

# Qu'est-ce qu'un programme ?

Wikipedia :

*« Un programme informatique est un ensemble d'opérations destinées à être exécutées par un ordinateur. »*

- C'est la description d'une méthode de résolution d'un problème : une suite d'instructions qui traitent les données du problème à résoudre, jusqu'à aboutir à une solution.

Il existe plusieurs catégories de programmes :

- **Programme source** (langage de programmation).  
Compilable ( $\Rightarrow$  binaire) (C, fortran) ou interprétable (Java, Python).
- **Programme binaire** (langage machine).  
exécutable par un microprocesseur.

# Exécution d'un programme

- Les traitements décrits par les instructions sont appliqués aux données (**entrées**),
- Les données ainsi transformées permettent d'aboutir aux solutions recherchées (**sorties**).

Le système d'exploitation<sup>1</sup> « démarre » un programme binaire :  
il ordonne au processeur d'exécuter les instructions du programme.

Par exemple, lorsqu'on double-clique sur une icône, ou bien lorsqu'on tape le nom du fichier exécutable en ligne de commande.

---

1. (windows, macos, ios, linux, android,...)

## Qu'est-ce qu'un programme (séquentiel) ?

Une recette, comme en cuisine. Une étape après l'autre.

- 1 Pelez et hachez finement 1/4 oignon(s).
- 2 Dans un bol :
  - 1 battez 2 oeufs entiers
  - 2 ajoutez 1 branche de basilic ciselé
  - 3 ajoutez 1/4 cube de bouillon de volaille.
  - 4 ajoutez sel et poivre à votre goût
  - 5 mélangez bien au fouet.
- 3 Dans une poêle :
  - 1 faites chauffer l'huile d'olive
  - 2 faites revenir les oignons hachés pendant 2 min, en remuant
- 4 Versez dans la poêle les oeufs et le fromage râpé laissez cuire 7 min.

# Écrire une recette $\neq$ faire la cuisine

- Une recette écrite par un auteur :
  - ▶ *est* un morceau de texte
- Une exécution par un cuisinier :
  - ▶ lit la recette
  - ▶ réalise les actions décrites



# Écrire une recette $\neq$ faire la cuisine

- Un programme écrit par un programmeur :
  - ▶ est un morceau de texte
- Une exécution par un processeur :
  - ▶ lit le programme
  - ▶ réalise les actions décrites
- 1 programme,  $\infty$  exécutions *différentes*
  - ▶ *paramètres* différents
  - ▶ *entrées* différentes (clavier, souris, fichier, etc)

# 1 programme, $\infty$ exécutions *différentes*

Ne pas confondre :

- Le **programmeur** qui *programme* :
  - ▶ *prévoit chaque* cas possibles à l'avance
  - ▶ décrit la (sous-)recette à suivre dans chacun des cas
- Le **processeur** qui *exécute*
  - ▶ ne fait que suivre la « recette » pas à pas
  - ▶ ne fera pas tous les cas lors d'*une* exécution

# Exemple de programme machine (hexadécimal)

```
00000000 fc 31 c0 8e c0 8e d8 8e d0 bc 00 7c 89 e6 bf 00 |.1.....|...|
00000010 06 b9 00 01 f3 a5 89 fd b1 08 f3 ab fe 45 f2 e9 |.....E..|
00000020 00 8a f6 46 bb 20 75 08 84 d2 78 07 80 4e bb 40 |...F. u...x..N.@|
00000030 8a 56 ba 88 56 00 e8 fc 00 52 bb c2 07 31 d2 88 |.V..V...R...1..|
00000040 6f fc 0f a3 56 bb 73 19 8a 07 bf 87 07 b1 03 f2 |o...V.s.....|
00000050 ae 74 0e b1 0b f2 ae 83 c7 09 8a 0d 01 cf e8 c5 |.t.....|
00000060 00 42 80 c3 10 73 d8 58 2c 7f 3a 06 75 04 72 05 |.B...s.X...;u.r.|
00000070 48 74 0d 30 c0 04 b0 88 46 b8 bf b2 07 e8 a6 00 |Ht.0...F.....|
00000080 be 7b 07 e8 b2 00 8a 56 b9 4e e8 8e 00 eb 05 b0 |.{...V.N.....|
00000090 07 e8 b0 00 30 e4 cd 1a 89 d7 03 7e bc b4 01 cd |...0.....~...|
000000a0 16 75 0d 30 e4 cd 1a 39 fa 72 f2 8a 46 b9 eb 16 |.u.0...9.r..F...|
000000b0 30 e4 cd 16 88 e0 3c 1c 74 f1 2c 3b 3c 04 76 06 |0....<.t.;<.v.|
000000c0 2c c7 3c 04 77 c9 98 0f a3 46 0c 73 c2 88 46 b9 |;.<.w...F.s..F.|
000000d0 be 00 08 8a 14 89 f3 3c 04 9c 74 0a c0 e0 04 05 |...<.t.....|
000000e0 be 07 93 c6 07 80 53 f6 46 bb 40 75 08 bb 00 06 |.....S.F.@u...|
000000f0 b4 03 e8 59 00 5e 9d 75 06 8a 56 b8 80 ea 30 bb |...Y.^..u..V...0.|
00000100 00 7c b4 02 e8 47 00 72 86 81 bf fe 01 55 aa 0f |.|...G.r...U...|
00000110 85 7c ff be 85 07 e8 19 00 ff e3 b0 46 e8 24 00 |.|.....F.$..|
00000120 b0 31 00 d0 eb 17 0f ab 56 0c be 78 07 e8 eb ff |.1.....V..x...|
00000130 89 fe e8 03 00 be 85 07 ac a8 80 75 05 e8 04 00 |.....u...|
00000140 eb f6 24 7f 53 bb 07 00 b4 0e cd 10 5b c3 8a 74 |..$.S.....[.t|
00000150 01 8b 4c 02 b0 01 56 89 e7 f6 46 bb 80 74 13 66 |..L...V...F..f|
00000160 6a 00 66 ff 74 08 06 53 6a 01 6a 10 89 e6 48 80 |j.f.t..Sj.j...H.|
00000170 cc 40 cd 13 89 fc 5e c3 20 20 a0 0a 44 65 66 61 |.|e...^...Defa|
00000180 75 6c 74 3a a0 0d 8a 00 05 0f 01 06 07 0b 0c 0e |ult:.....?|
00000190 83 a5 a6 a9 0d 0c 0b 0a 09 08 0a 0e 11 10 01 3f |.....?|
000001a0 bf 44 4f d3 4c 69 6e 75 f8 46 72 65 65 42 53 c4 |.D0.Linu.FreeBS.|
000001b0 66 bb 44 72 69 76 65 20 00 00 80 8f b6 00 00 00 |f.Drive .....|
000001c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*
000001f0 00 00 00 00 00 00 00 00 00 00 00 00 55 aa |.....U..|
00000200
```

# Exemple de programme machine (assembleur)

```
.data
UnNom :
.long 43,54,32,76 /* 4 entiers. */
.globl _start
_start:
movl $5, %eax /* EAX nombre d'entiers restant à additionner */
movl $0, %ebx /* EBX va contenir la somme de ces entiers */
movl $UnNom, %ecx /* ECX << pointe >> sur l'élément à additionner */
top:
addl (%ecx), %ebx /* Additionne EBX ECX, résultat dans EBX */
addl $4, %ecx /* Déplace le <<pointeur>> sur le suivant */
decl %eax /* Décrémente le compteur EAX */
jnz top /* Si EAX non nul <<sauter>> à top: */
done:
movl %ebx, UnAutre /* sinon, le resultat est stocké */
movl $0,%ebx /* Ces instructions permettent d'invoquer de */
movl $1,%eax /* terminer l'exécution d'un programme */
int $0x80 /* assembleur et sont indispensable */
```

Commentaire (sans effet)

Début des instructions

## Exemple de programme C

```
#include <stdio.h>
void main(void)
{
    int i;
    char prenom[30];
    FILE *FICHIER1;
    FICHIER1=fopen("/home/chr/premierfichier.txt","w");
    for(i=1;i<=10;i++)
        {
            printf("Rentrez un prénom :\n");
            scanf("%s",prenom);
            fprintf(FICHIER1,"%s\n",prenom);
        }
    fclose(FICHIER1);
}
```

# Les langages de programmation

**Les langages de haut niveau :** (Ex : C, Ada, Pascal, Cobol, Java, OCaml, Python).

- fournissent des constructions sophistiquées qui facilitent l'écriture des programmes.
- sont compréhensibles par les humains, mais pas directement exécutables par les machines.
- $\Rightarrow$  Traduction en langage machine avant son exécution.

**Les langages de bas niveau :** (langages cibles ou natifs)

- instructions propres à chaque machine (SPARC/Sun, Intel/PC, etc).
- codés en binaire et directement exécutables par chaque machine.

# Description d'un langage de haut niveau

**Les types des données** : utilisés pour modéliser les données.

**Exemple** : Le type `int` en C sert à modéliser les nombres entiers.

**La syntaxe** : règles de formation textuelle des instructions.

**Exemple** : pour écrire l'expression mathématique  $1 \leq x \leq 7$ , une syntaxe possible en C est : `1 <= x && x <= 7`.

**La sémantique** : règles qui précisent "le sens" des constructions syntaxiques à l'exécution et la cohérence entre types.

**Exemples** :

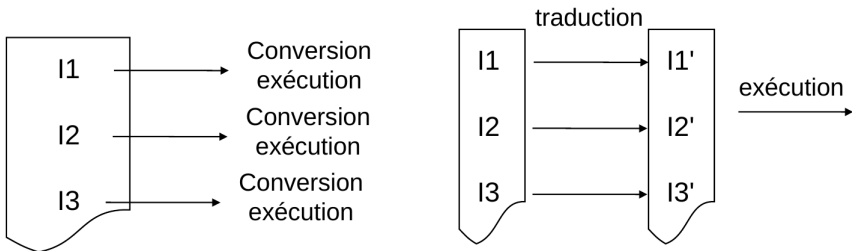
- `4+3*2` équivaut à la valeur entière `10`
- `"bonjour"*2` est correct syntaxiquement, mais non sémantiquement :  
\* ne peut pas s'appliquer à une chaîne de caractères

# Compilation vs Interprétation

Il faut **traduire** le langage de haut niveau vers le langage machine spécifique à un processeur  $\Rightarrow$  il peut lire les instructions et les exécuter.

## 2 méthodes :

- **Compilation** : traduction de toutes les instructions, puis exécution de la traduction : le résultat dépend de l'ordinateur visé
- **Interprétation** : conversion et exécution de chaque instruction les unes derrière les autres : plus flexible mais plus lent.





# Un premier programme C

# Structure d'un programme en C

Un programme contient C au moins un fichier principal contenant une fonction `main` qui reçoit la suite d'instructions à exécuter.

```
#include "inout.h"
int main(int nargs, char **args)
{
    ecrireString("Hello world!\n");
    ecrireString("Bye world!\n");
}
```

Permet `ecrireString`, etc

Fonction principale

Début du programme

Première instruction

# En ligne de commande (unix,gcc)

Source

```
inout.c
monprogramme.c
"Compiler"
```

```
inout.o
monprogramme.o
```

monexe

bibliothèque maison (lecture au clavier)

Écrit par vous

```
gcc -c inout.c          -> inout.o
gcc -c preprogramme.c  -> preprogramme.o
gcc inout.o preprogramme.o -o monexe  -> monexe
```

"construire un exécutable" (link)

"nom de l'exécutable"

- répertoire courant : `inout.c`, `inout.h` et `preprogramme.c`
- link : tous les fichiers `.o` d'un seul coup, ordre important
- raccourci quand un seul fichier : `gcc monpogramme.c -o monexe`
- Exécution : `./monexe`

# Les variables

## Comment mémoriser les opérations effectués ?

Considérons les fonctions :

- `lireInt();` qui lit un entier tapé au clavier, on dit qu'elle **retourne** l'entier tapé.
- `ecrireInt(int a);` qui affiche à l'écran l'entier **a** passé en paramètre.

On souhaite lire un entier **n** au clavier, puis afficher la valeur de  $n + n^2$ .

- `ecrireInt(lireInt()+lireInt())` fait 2 lectures clavier.
- comment faire pour utiliser plusieurs fois le résultat du premier `lireInt()` ?

⇒ Stockage dans une **variable** de la valeur retournée par (du résultat) `lireInt()` pour l'utiliser plus tard.

## Qu'est ce qu'une variable ?

Une **variable** est une portion de mémoire qui a un nom (en C : n'importe quel nom commençant par une lettre).

Elle a les propriétés suivantes :

- elle doit être suffisamment grande pour contenir la donnée (ex : contenir un `int`)
- elle change de valeur au cours de l'exécution du programme

Il est nécessaire de la **déclarer** avant de l'utiliser.

Les opérations possibles sur les variables sont :

- **l'affectation** : remplacer sa valeur courante par une autre valeur.
- **l'utilisation** : utiliser sa valeur courante dans une instruction.

# Les variable : exemple en C

type

nom

- *Déclaration* :

```
int n ;  
int x, m ;
```

- *Affectation* :

```
n = 12 ;  
x = 3+4 ;  
m = lireInt()+7 ;
```

- ▶ signe « égal » (=) ne signifie pas égalité (voir != !)
- ▶ `x = 12;` signifie « mettre la valeur 12 dans la variable `x` ».

- *Utilisation* :

```
ecrireInt(n+8+g(n+3));  
f(n + n*n + x * m);  
x = g(n + 3 , m - 7);
```

## Variable – Affectation (1/3)

```
x = g(n + 3, m - 7);
```



« Receptacle »

« Valeur »



**pas la même signification à gauche ou à droite de l'affectation**

### Règle

Le nom d'une variable désigne toujours sa valeur sauf à gauche de l'affectation.



## Variable – Affectation (2/3)

Déroulement (à l'exécution) de l'affectation :

```
x = g(n +3 , m - 7);
```

- partie de droite (ici `g(n +3 , m - 7)`) exécutée (donne un **résultat**)
- *ensuite* le **résultat** de la partie droite est écrit dans la variable de gauche (`x`) (efface la valeur précédente)

Un exemple de modification de la valeur d'une variable :

```
int n = 4;  
ecrireInt(n); // ? qu'affiche ceci  
n = n + 3;  
ecrireInt(n); // ? qu'affiche ceci
```

## Variable– Affectation (3/3)

L'instruction d'affectation :

```
x = f(12) + 7 - g(4)
```

S'exécute de la manière suivante :

- 1 Évaluation à droite du « = » : `f(12) + 7 - g(4)`
- 2 Valeur obtenue stockée dans `x` (valeur précédente écrasée).

**Question** : Que fait l'instruction `x = x + 2;` ?

# Les types de données

# Notion de type de données

Toutes les variables ont un **type**, il définit :

- la taille de l'espace mémoire réservé pour celle-ci.
- l'ensemble des valeurs que peut prendre un objet informatique ainsi que les opérations permises sur ces valeurs.

**Un objet informatique** est toute entité à laquelle un type est associé (variable, constante, fonction).

**Exemple : le type entier `int` :**

- Ensemble de valeurs :  $\mathbb{N}$
- Ensemble d'opérations sur ces valeurs (`+`, `-`, `*`, `/`, `==`, ...)

## Intérêt du concept de type

Fournir à la machine les **informations** nécessaires pour qu'elle soit en mesure de **vérifier que l'utilisation** des objets d'un programme est bien **conforme** à l'intention préalable du programmeur :

- Lorsque nous déclarons un **objet (variable, constante, fonction, ...)**, nous spécifions les contraintes de son utilisation, l'ensemble des valeurs qui pourront lui être associées et ainsi le **cadre légitime de son utilisation**.
- L'utilisation de cet objet sera précédée d'une **vérification effectuée par le compilateur**. La valeur qu'il prend appartient-elle à cet ensemble ? Son utilisation est-elle conforme aux contraintes spécifiées ?

**Le concept de type permet d'établir un lien sémantique entre la déclaration et l'utilisation d'un objet.**

## Les types primitifs

Type	Ensemble des valeurs	Taille
byte	$[-128, \dots, 127]([-2^7, \dots, 2^7 - 1])$	1 octet
short	$[-32768, \dots, 32767]([-2^{15}, \dots, 2^{15} - 1])$	2 octets
int	$[-2^{31}, \dots, 2^{31} - 1]$	4 octets
long	$[-2^{63}, \dots, 2^{63} - 1]$	8 octets
float	réels simple précision (7 chiffres significatifs)	4 octets
double	réels double précision (15 chiffres significatifs)	8 octets
char	les caractères	2 octets

# Opérateurs et expressions

- Un **opérateur** permet de réaliser des calculs sur une ou plusieurs valeurs appelées **opérandes**.
- Il existe des opérateurs unaires (une seule opérande), binaires et ternaires (ex :  $x++$ ,  $9 * x$ )
- Les **expressions** sont des constructions décrivant des opérations à l'aide d'opérateurs et d'opérandes
- Les opérandes d'une expression peuvent être des constantes, des variables valuées ou des appels de fonction :
  - ▶  $(34 + \min(x,y))$  où  $x$  et  $y$  sont les paramètres de la fonction  $\min$ .
  - ▶  $2 > 5$  est une expression booléenne dont le résultat est faux
  - ▶  $!(2 > 5)$  est une expression booléenne lue : non 2 plus grand que 5
- Une expression correspond toujours à la valeur qu'elle calcule

# Opérateurs arithmétiques

- **Opérateurs sur les types entiers** (byte, short, int, long)
  - ▶ +, -, \*, /, %
  - ▶ Exemples :  $9/4 = 2$ ,  $9\%4 = 1$
- **Opérateurs sur les types réels** (double, float)
  - ▶ +, -, \*, /
  - ▶ Exemple :  $9/4 = 2,25$
- **Opérateurs relationnels**
  - ▶ <, >, <=, >=, ==, !=
  - ▶ Exemples :  $5 > x$ ,  $a == 'b'$ ,  $(a != 0) \ \&\& \ (a <= 100)$



## Précédence des opérateurs (1/2)

Les expressions composées de plusieurs opérateurs sont évaluées de la **gauche vers la droite**, selon des **règles de priorité** indiquant la priorité des opérateurs les uns par rapport aux autres.

- $2 + 3 * 4 \Rightarrow$  équivaut à  $2 + (3*4)$   
 $\Rightarrow$  s'évalue en 14 et non pas en 20
- $2 + 3 > 4 \Rightarrow$  s'évalue en vrai
- $2 + (3 > 7) \Rightarrow$  produit une erreur

## Précédence des opérateurs (2/2)

De la plus haute à la plus basse priorité :

+ , -  
!  
\* / %  
+ -  
< <= > >=  
== !=  
&&  
||

# Les instructions et les expressions

# Qu'est ce qu'une instruction ?

```
ecrireString("Hello world\n");
```

## Une *instruction*

- morceau de programme/recette = morceau de texte,
- se termine (en C, C++, C) par un point-virgule : ;
- opération *réalisée* à l'exécution,

## Exemples :

- « battez 2 oeufs entiers »
- ```
ecrireString("Toto\n");
```
- ```
x = y + 3;
```
- ```
return (3);
```

## Contre-exemples :

- « 2 oeufs »
- ```
"Toto\n"
```
- ```
y+3
```
- ```
(3)
```

# Qu'est ce qu'une expression ?

"Hello world\n"

45

2 + 7 \* 0xFF

x

2 + x \* 0xFF

## Une *expression*

- morceau de programme/recette = morceau de texte,
- une *partie* d'une instruction
- qui correspondra à une *valeur* à l'exécution,

## Contre-exemples :

- « 2 oeufs »
- "Toto\n"
- y+3
- (3)

## Exemples :

- « battez 2 oeufs entiers »
- écrireString("Toto\n");
- x = y + 3;
- return (3);

# Instruction VS expression

Quelques règles :

Instruction	Expression
se termine par ;	fait partie d'une instruction
aura un <i>effet</i> à l'exécution	aura une <i>valeur</i> à l'exécution
n'a pas de type ( <code>void</code> )	a un type ( <code>int</code> , <code>double</code> , etc)

En C :

- les expressions sont parfois aussi des instructions.
- retournent une valeurs ET ont un effet.
- ex : `x = 3`
  - ▶ effet : met la valeur 3 dans la variable `x`
  - ▶ valeur : 3.
  - ▶ code bizarre : `x = (y = f(x,y))` à éviter

# Appeler un sous-programme

# Les sous-programmes

Un sous-programme est un « bout de programme »

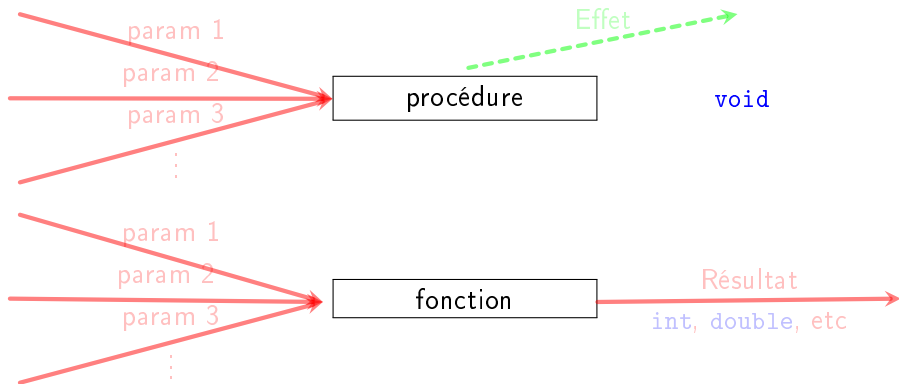
- qui utilise des *paramètres* (ou arguments)
- qu'on peut appeler autant qu'on veut *avec des paramètres différents*
- qui peut avoir des *effets* (modif. écran ou mémoire, etc)
- qui peut avoir un *résultat* (valeur de « retour »)

Il peut être :

- écrit par soi-même
- ou par un autre programmeur
- code source connu/inconnu



## Sous-programme : procédures VS fonction



procédure = instruction/expression ?      fonction = instruction/expression ?

# Appel de procédure

```
ecrireDate();
```

Appel de la procédure `ecrireDate`

- 1 Séparez les feuilles de la salade
  - 2 Lavez les feuilles
  - 3 **Faites une vinaigrette, page 12.**
  - 4 égouttez les feuilles
  - 5 ...
- Appel de la procédure `vinaigrette`

## Appel de procédure I – exécution

Salade verte :

1. Séparez les feuilles
2. Lavez les feuilles
3. **Faites une vinaigrette**
9. égouttez les feuilles
10. ...

Vinaigrette :

4. versez l'huile dans un bol
5. versez le vinaigre
6. versez la moutarde
7. battre jusqu'à émulsion
8. salez poivrez

- 1 *arrête* l'exécution de la recette courante
- 2 démarre l'autre recette **avec le paramètre réel**
- 3 autre recette finie  $\Rightarrow$  *redémarre la recette courante*

## Deuxième programme (procédure `pause`)

procédure `pause()` attend une pression sur « entrée ».

procédure `ecrireDate()` écrit la date à l'écran.

```
#include "inout.h"

int main(int nargs, char **args)
{
    ecrireString("Hello world! Please press enter\n");
    pause();
    ecrireDate();
    ecrireString("Bye world!\n");
}
```

## Appel de procédure II – Paramètres

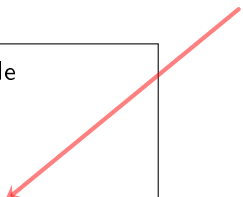
```
ecrireString("Hello world\n");
```

```
ecrireInt(12);
```

paramètres



### Omelette salade

- 1 Séparez les feuilles de la salade
  - 2 Lavez les feuilles
  - 3 Faites une vinaigrette (p.12)
  - 4 **Faites une omelette pour 4 (p.23)**
  - 5 égouttez les feuilles
  - 6 ...
- 

# Paramètre de procédures

## Omelette pour 1 personne(s)

- 1 Pelez et hachez finement 1/4 oignon(s)
- 2 Dans un bol :
  - 1 battez 2 oeufs entiers
  - 2 ajoutez 1 branche(s) de basilic ciselé
  - 3 ajoutez 1/4 cube(s) de bouillon de volaille.
  - 4 ajoutez sel et poivre à votre goût
  - 5 mélangez bien au fouet.
- 3 Dans une poêle :
  - 1 faites chauffer l'huile d'olive
  - 2 faites revenir les oignons hachés pendant 2 min, en remuant
- 4 Versez dans la poêle les oeufs et le fromage râpé laissez cuire 7 min.

# Paramètre de procédures

⇒ Paramètre  $n$  (formel)

Omelette pour  $n$  personne(s)

- 1 Pelez et hachez finement  $n/4$  oignon(s)
- 2 Dans un bol :
  - 1 battez  $2 \times n$  oeufs entiers
  - 2 ajoutez  $n$  branche(s) de basilic ciselé
  - 3 ajoutez  $n/4$  cube(s) de bouillon de volaille.
  - 4 ajoutez sel et poivre à votre goût
  - 5 mélangez bien au fouet.
- 3 Dans une poêle :
  - 1 faites chauffer l'huile d'olive
  - 2 faites revenir les oignons hachés pendant 2 min, en remuant
- 4 Versez dans la poêle les oeufs et le fromage râpé laissez cuire 7 min.

# Paramètre de procédures

Omelette pour 3 ← paramètre réel

Omelette pour 3 personne(s)

- 1 Pelez et hachez finement 3/4 oignon(s)
- 2 Dans un bol :
  - 1 battez 2×3 oeufs entiers
  - 2 ajoutez 3 branche(s) de basilic ciselé
  - 3 ajoutez 3/4 cube(s) de bouillon de volaille.
  - 4 ajoutez sel et poivre à votre goût
  - 5 mélangez bien au fouet.
- 3 Dans une poêle :
  - 1 faites chauffer l'huile d'olive
  - 2 faites revenir les oignons hachés pendant 2 min, en remuant
- 4 Versez dans la poêle les oeufs et le fromage râpé laissez cuire 7 min.



# Paramètre de procédures

## Omelette pour 4

### Omelette pour 4 personne(s)

- 1 Pelez et hachez finement 4/4 oignon(s)
- 2 Dans un bol :
  - 1 battez 2×4 oeufs entiers
  - 2 ajoutez 4 branche(s) de basilic ciselé
  - 3 ajoutez 4/4 cube(s) de bouillon de volaille.
  - 4 ajoutez sel et poivre à votre goût
  - 5 mélangez bien au fouet.
- 3 Dans une poêle :
  - 1 faites chauffer l'huile d'olive
  - 2 faites revenir les oignons hachés pendant 2 min, en remuant
- 4 Versez dans la poêle les oeufs et le fromage râpé laissez cuire 7 min.

## Troisième programme

```
#include "inout.h"

int main(int nargs, char **args) {
    ecrireString("\nHello world!\n");
    pause();
    ecrireInt(12);           Écrit un entier à l'écran
    ecrireString("\nBye world!\n");
}
```

## Quatrième programme

```
#include "inout.h"

int main(int nargs, char **args) {
    ecrireString("\nHello world!\n");
    pause();
    ecrireInt(12+5); entier
    ecrireString("\nBye world!\n");
}
```

# Les fonctions

Considérons la fonction `heureActuelle();`

Elle « retourne » le numéro de l'heure actuelle (entre 0 et 23)

- `heureActuelle()` **équivalent à un entier**
- `heureActuelle() + 2` bloque l'addition jusqu'à son retour (démonstration)

Pour l'utiliser, il faut l'inclure dans une instruction.

ex : `ecrireInt(heureActuelle());`

`ecrireInt` attend le retour de la fonction `heureActuelle()`

# Appel de fonction

Salade verte pour 3 :

1. Séparez les feuilles de la salade
2. Lavez les feuilles<sup>3</sup>
3. égouttez les feuilles
4. Faites une vinaigrette pour 3 et versez la
10. ...

Vinaigrette pour 3 :

5. versez 3 cuillère d'huile dans un bol
6. versez 3/3 cuillères de vinaigre
7. versez 3/3 cuillères de moutarde
8. battre jusqu'à émulsion
9. salez poivrez

résultat =  
vinaigr.

- ❶ *arrête* l'exécution de la recette courante : **instruction 4 pas finie !**
- ❷ démarre l'autre recette **avec le paramètre réel**
- ❸ autre recette finie  $\Rightarrow$  *instruction 4. redémarre en utilisant la valeur retournée*

# Fonction et procédures utiles

```
lireInt();
```

- Attend que l'utilisateur tape un entier (ex : 123) + « entrée »
- « retourne » l'entier tapé
- `lireInt()` **équivalent à un entier**
- `ecrireInt(lireInt())` (démonstration)
- `ecrireInt(lireInt() + lireInt())` (démonstration)

## Erreur extrêmement courante



Répétez en cœur :

« Prendre en paramètre n'est pas lire au clavier »

« Retourner un résultat n'est pas afficher à l'écran »

