Algorithmique et programmation

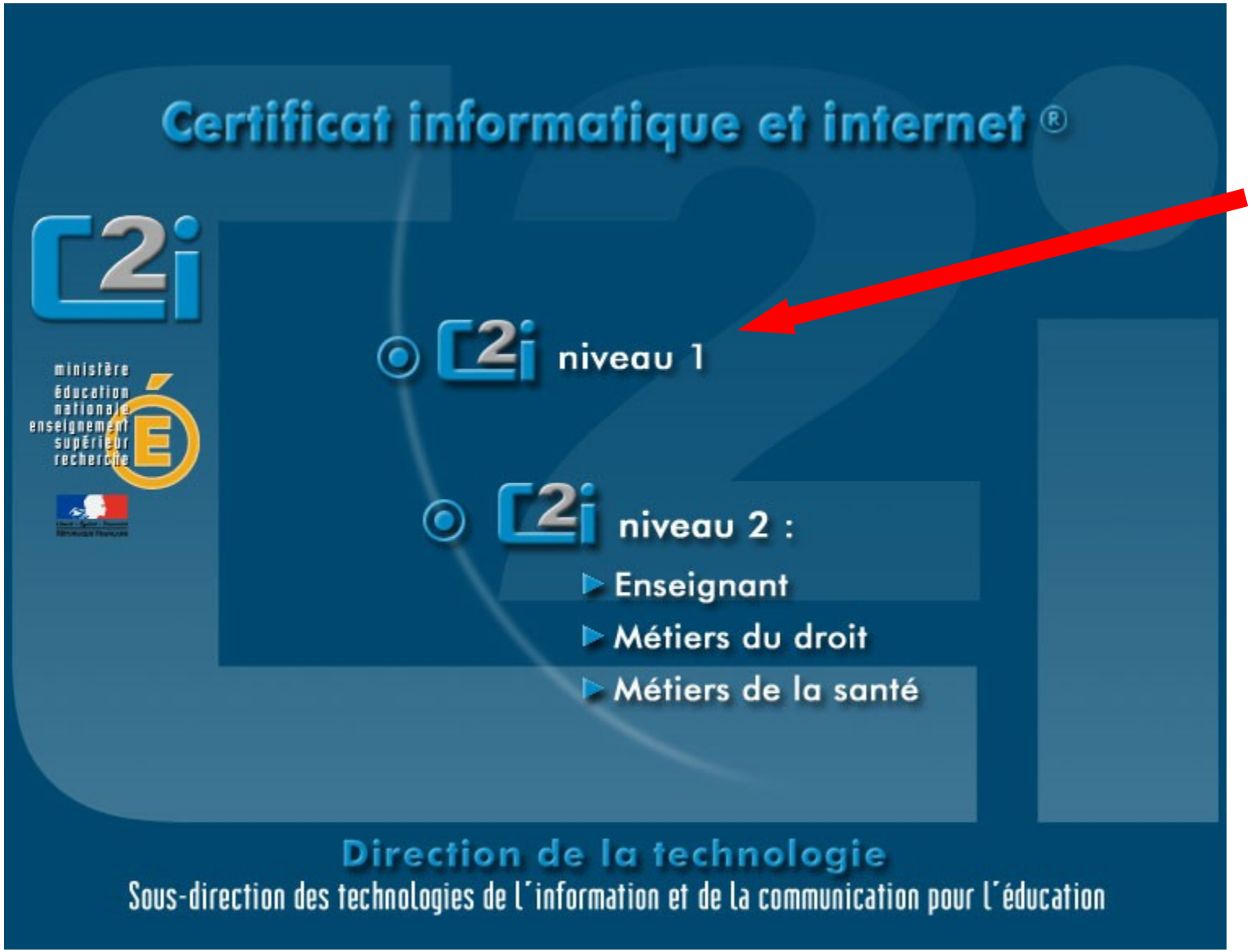
–**cours:**

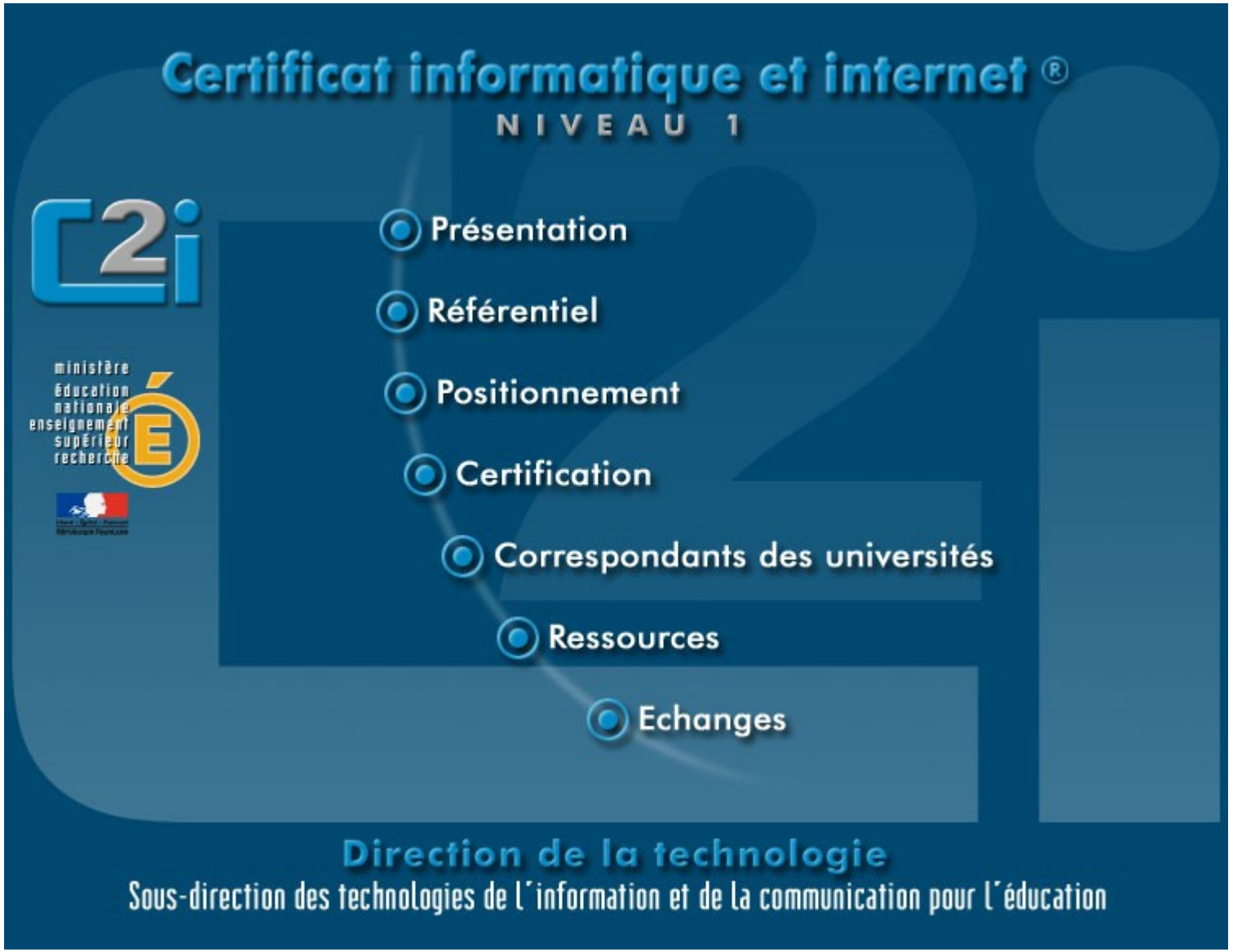
- Algorithmique et langage C
- *C2i (Certificat Informatique et Internet, niveau 1)*
  Autoformation bientôt disponible ?

–**Travaux dirigés:**

- préparation des travaux dirigés:
  libre service, disquette, etc...
- contrôle continu:
  2 ou 3 contrôles écrits en TD
  1 note de participation (préparations)
- note finale: (CC+partiel)/2
- C2i : *TD1+ pratiques annexes*

Le site internet : http://c2i.education.fr




Certificat informatique et internet ®
NIVEAU 1
C2i
ministère éducation nationale enseignement supérieur recherche
Présentation
Référentiel
Positionnement
Certification
Correspondants des universités
Ressources
Echanges
Direction de la technologie
Sous-direction des technologies de l'information et de la communication pour l'éducation

Algorithmique et programmation

Bibliographie:

Algorithmique:

- «Algorithmes et structures de données»
  de Niklaus Wirth Eyrolles
- «Types de données et algorithmes»
  de Christine Froidevaux, Marie-Claude Gaudel, Michèle Soria
  Mc Graw-Hill

etc...

Langage C:

- «Le langage C»
  de Brian W. Kernighan & Denis M. Ritchie Dunod
- voir documents en ligne http://infolicencemass.univ-paris1.fr

Algorithmique et programmation

Avoir accès au langage C:

- Système Linux...
- Knoppix
- Windows : Minimalist GNU for windows 4.1

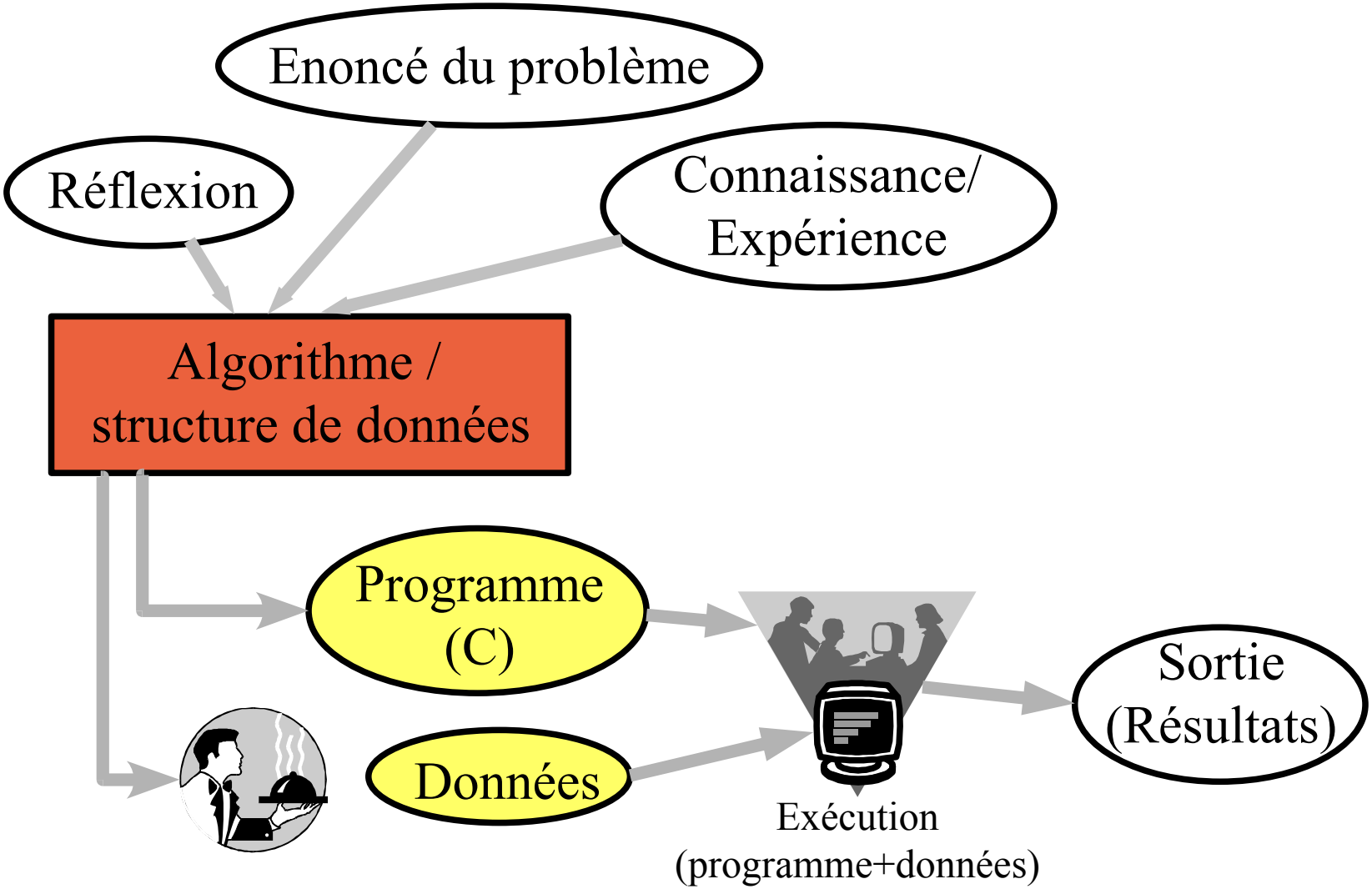
  MinGW-4.1.0.exe pris sur http://www.mingw.org/download.shtml

Salles de libre service: 4ème étage

+ *clé USB ou disquette...*

Algorithmique et programmation

Les différents aspects



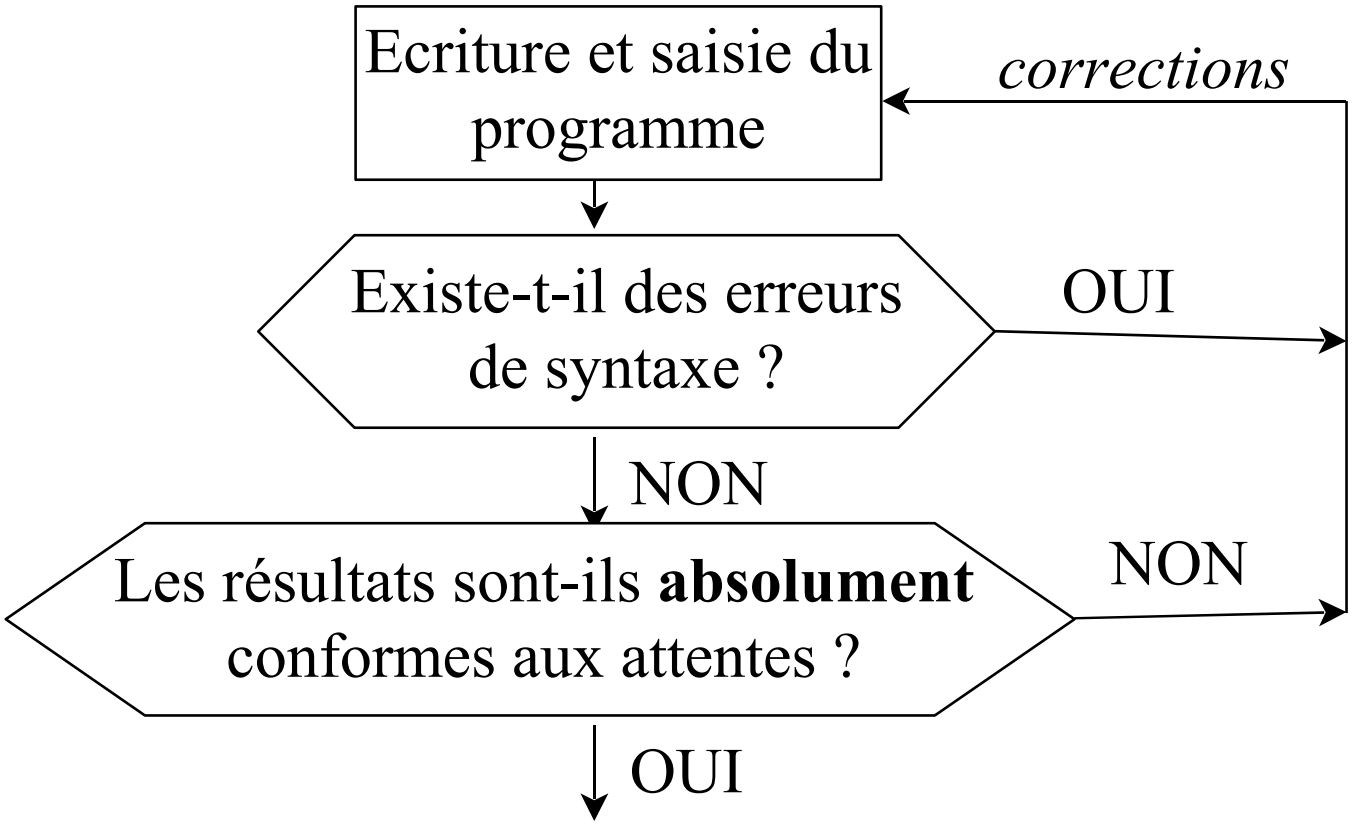
Algorithmique et programmation

Algorithme :

Ensemble fini de règles déterminées servant à résoudre un problème au moyen d'un nombre fini d'opérations.

Algorithmique et programmation

Réalisation et mise au point du programme



Algorithmique et programmation

***Exemple* : faire la somme de n nombres.**

Technique de base :

- supposer qu'on a fait la somme des i-1 premiers nombres
- Examiner le traitement à faire pour avoir la somme des i premiers nombres à partir de la somme des i-1 premiers nombres:

  - lire le i-ème nombre,
  - l'ajouter à la somme "courante".

- Il ne "reste plus" qu'à répéter cela le bon nombre de fois, en initialisant la somme "courante" à une valeur adaptée au problème.

Algorithmique et programmation

***Exemple* : faire la somme de n nombres (suite et fin)**

description de l'algorithme complet :

- Somme ← 0
- pour i variant de 1 à n faire :
  - lire le i-ème nombre
  - ajouter ce nombre à la Somme
- la Somme est disponible

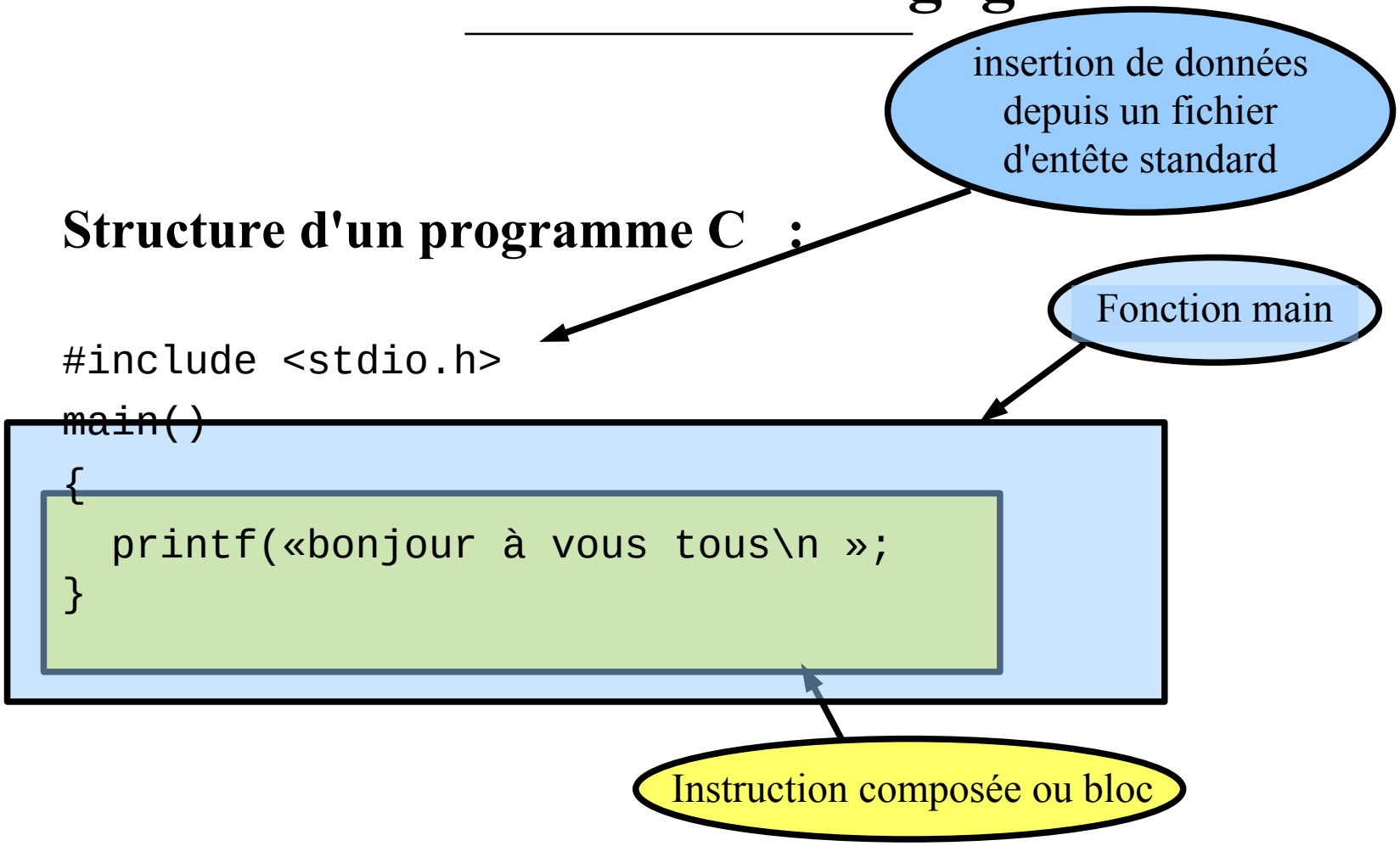
Introduction au langage C

Exemple de programme C :

```
#include <stdio.h>
main()
{
  printf(«bonjour à vous tous\n »;
}
```

Introduction au langage C

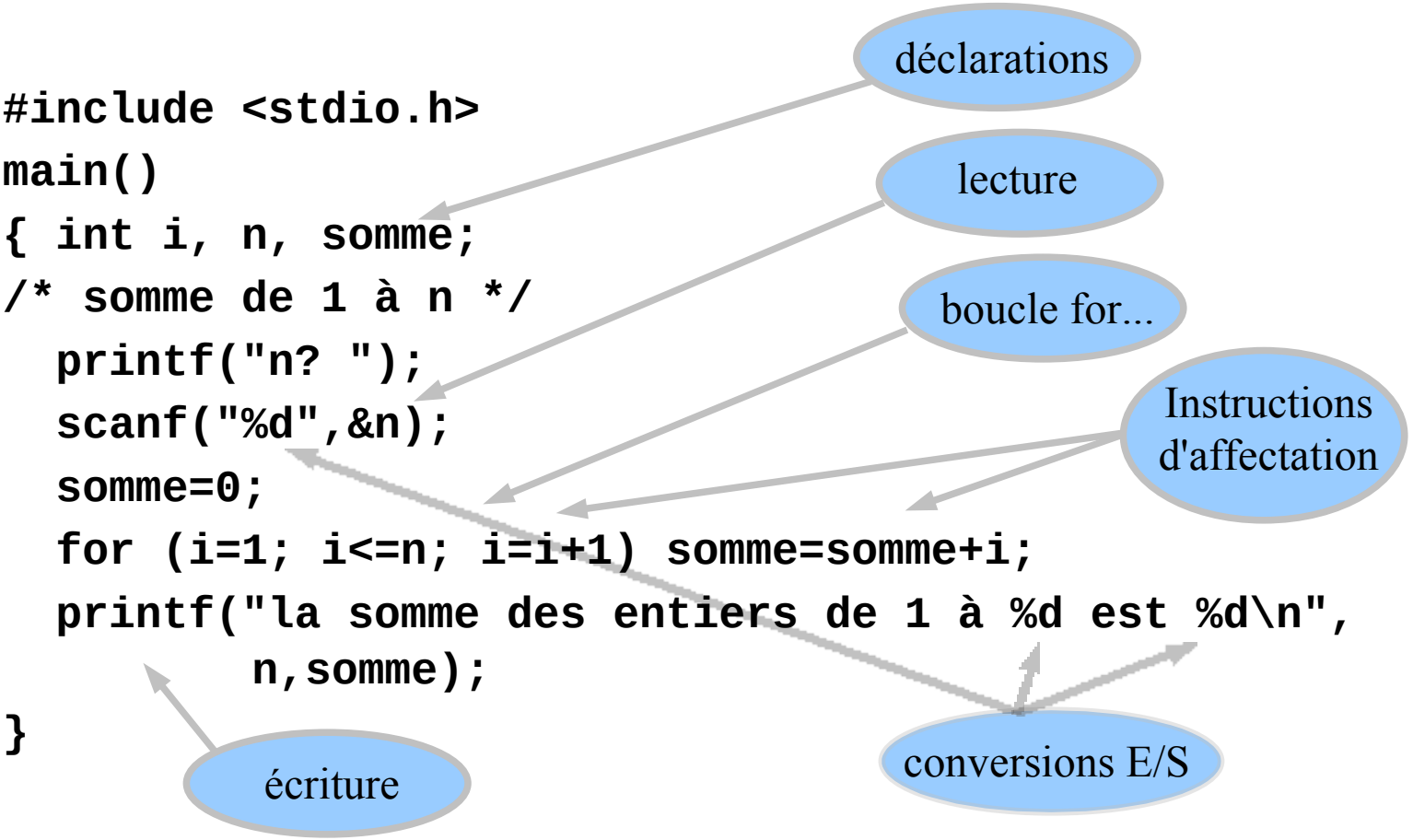
Structure d'un programme C :



Introduction au langage C

Exemple 2 de programme C :

```
#include <stdio.h>
main()
{ int i, n, somme;
/* somme de 1 à n */
  printf("n? ");
  scanf("%d",&n);
  somme=0;
  for (i=1; i<=n; i=i+1) somme=somme+i;
  printf("la somme des entiers de 1 à %d est %d\n",
         n,somme);
}
```



Introduction au langage C

Les types de données de base :

int	nombre "entier"
char	caractère
float	nombre "réel"
double	nombre "réel double précision"

Qualificatifs pouvant modifier les types de base :

short int	**long** int	**long** double
unsigned char	**signed** char	

Introduction au langage C

Exemples de déclarations :

```
int i, j, bonjour_a_tous;
char c, c1;
float x;
double y, total_ttc;
long somme;
```

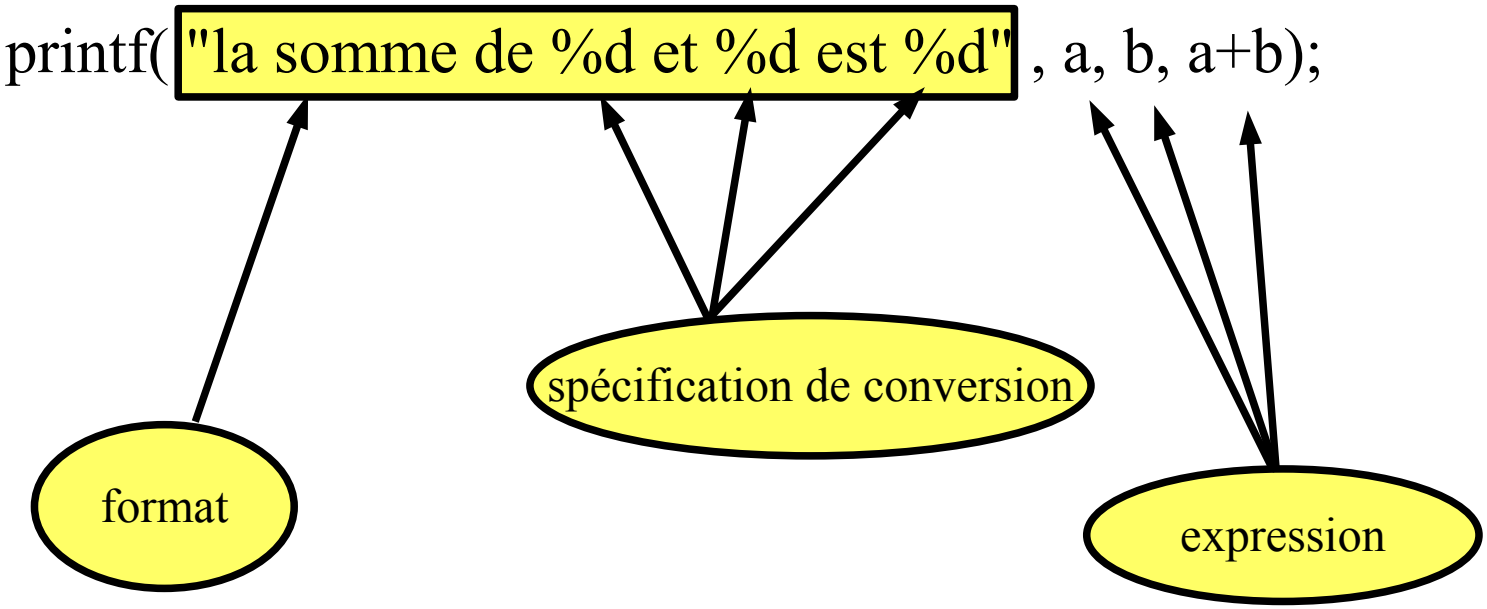
Syntaxe d'un nom de variable :

lettres, caractère souligné "_", chiffres.

commence par une lettre ou "_"

Introduction au langage C

Affichage : instruction printf



Introduction au langage C

Quelques caractères de contrôle:

\n	fin de ligne
\t	tabulation (horizontale)
\\	backslash
\'	apostrophe
\"	guillemet

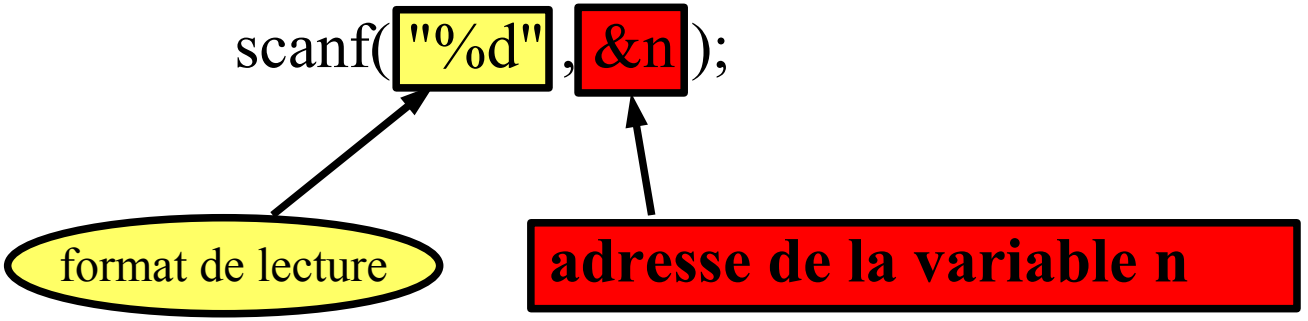
```
printf("la somme de %d et %d est %d\n", a, b, a+b);
```

Largeur d'affichage :

```
printf("la somme de %4d et %4d est %5d", a, b, a+b);
```

Introduction au langage C

lecture : instruction scanf

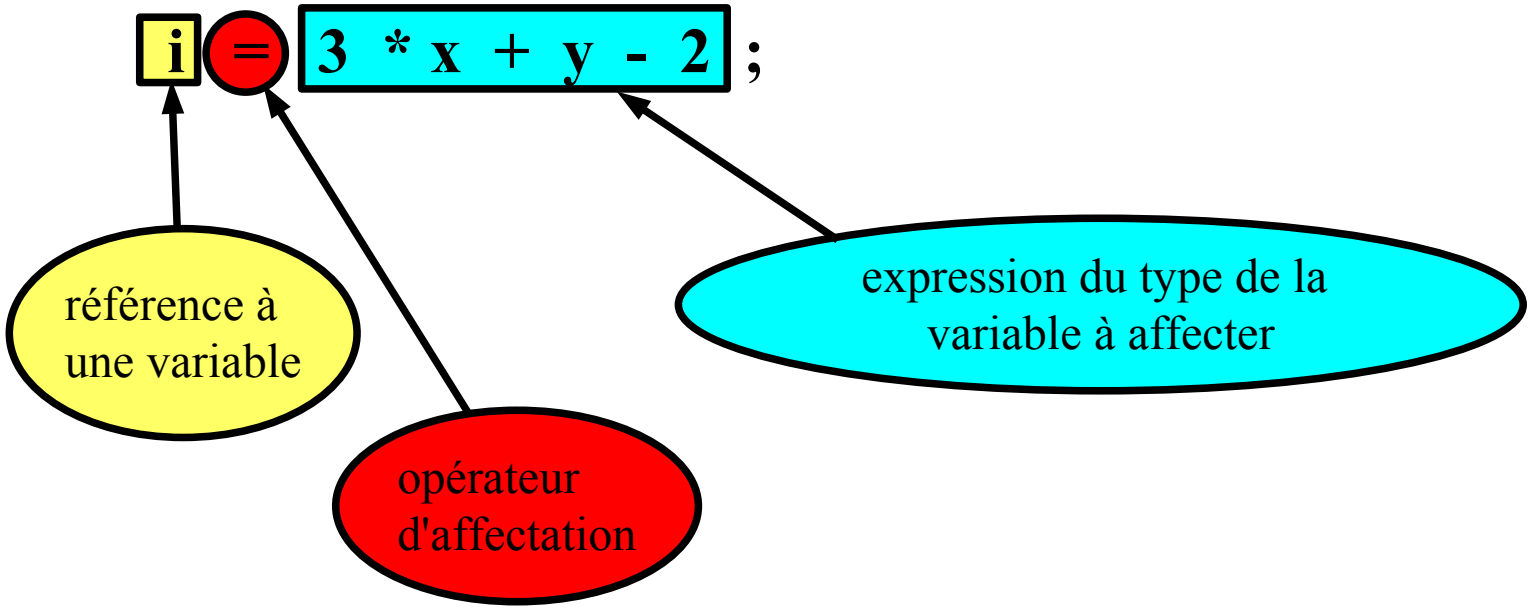


& : opérateur de prise d'adresse.

Règle : scanf, après le format de lecture, doit avoir la liste des adresses des variables à lire

Introduction au langage C

Instruction d'affectation:



1. l'expression de droite est calculée

2. la variable de gauche reçoit la valeur de l'expression

Introduction au langage C

Comment modifier la valeur d'une variable ?

```
somme = somme + x ;
produit = produit * k ;
truc = truc * (machin + 2) - qqc ;
```

"Simplifications" d'écriture du langage C:

```
somme += x ;
produit *= k;
```

D'une façon plus générale : **variable op= expression** **équivaut à** **variable = variable op expression**

avec op : **+ - * / %** ...pour commencer

Introduction au langage C

Opérateur	associativité
( )	de gauche à droite
! ++ -- + - * & (unaire)	*de droite à gauche*
* / %-	de gauche à droite
+ -	de gauche à droite
< <= > >=	de gauche à droite
= = ! =	de gauche à droite
&&	de gauche à droite
||	de gauche à droite
? :	*de droite à gauche*
= + = - = / = % =	*de droite à gauche*
,	de gauche à droite

Tableau limité aux opérateurs qui seront vus durant ce semestre.

Introduction au langage C

La répétition d'un même traitement:

traitement itératif, boucles

instruction while

instruction for

instruction do

Introduction au langage C

L'instruction while :

while (*expression*)

instruction

exemple :

```
...
S = 0;
while ( n > 0 )
{
   S = S + n;
   n -= 1;
}
...
```

Introduction au langage C

L'instruction for :

for (*expression1 ; expression2; expression3*)
instruction

exemple :

```
...
for(S = 0; n > 0; n-- )
  S = S + n;
...
```

Utilisées seules : n-- --n n-=1 n=n-1 sont équivalentes
n++ ++n n+=1 n=n+1 " "

Introduction au langage C

Equivalence des instructions while et for :

```
for (expression1 ; expression2; expression3)
    instruction
```

```
expression1 ;
while ( expression2)
   {
      instruction ;
      expression3 ;
   }
```

Introduction au langage C

Utilisation de l'instruction composée :

- **La syntaxe exige une seule instruction**
- **la logique du traitement en exige plusieurs**

```
{
    instr1 ;
      ....
    dernière_instruction ;
}
```

Introduction au langage C

Les complications arrivent :

- l'opérateur `++` ou `--` au sein d'une expression :

exemple:

```
x = 2 ;                    x = 2 ;
n = 3 ;                    n = 3 ;
x = x * n-- ;              x = x * --n ;
```

Que valent x et n après l'exécution des 2 instructions ?

n est incrémenté après son utilisation →

x ← 2 * 3
puis n ← 3 - 1

n est incrémenté avant son utilisation →

n ← 3 - 1
puis x ← 2 * 2

Introduction au langage C

- **les expressions en C :**
  - les expressions logiques :

    1) test logique:

    une expression est = 0 ⇒ false

    une expression est # 0 ⇒ true

exemple:

```
if (n % d)
{ .....
```

Introduction au langage C

- **les expressions en C :**
  - les expressions logiques :

2) valeur d'une expression logique:

false ⇒ **0**

true ⇒ **1**

exemple:

```
printf("%d\n",x > y);
```

Introduction au langage C

- **les expressions en C :**
  - les expressions logiques :

3) les opérateurs logiques et évaluation:

!	:	**négation**
& &	:	**et**
| |	:	**ou (inclusif)**

exemples:

```
(poids > 100) && (taille < 195)
!n
(a < b) && (c < d) && (e < f)
(a < b) || (c < d) || (e < f)
```

arrêt de l'évaluation dès que le résultat est acquis

Introduction au langage C

- **les expressions en C :**
  - l'opérateur virgule :

expr1, expr2, ... , expression-d'affectation

évaluation de gauche à droite

résultat: type & valeur de l'opérande de droite

exemple:

```
for(i=1,j=2;i<N;i++)...
```

Introduction au langage C

- **les expressions en C :**
  - l'opérateur = (affectation) :

expression-d'affectation:
expr-modifiable = expression-d'affectation

évaluation de droite à gauche

résultat: type & valeur de l'opérande de gauche

exemple:

```
a = x = y = 1;
```

remarque: = peut être remplacé par les autres opérateurs d'affectation : *= /= %= += -= ...

Introduction au langage C

L'instruction do :

```
do instruction
while (expression) ;
```

exemple1 :

```
s = 0; p = 0;
do s += ++p;
while (p < n);
```

exemple2 :

```
...
do
{ ...<instruction composée>
}
while (expression);
```

Introduction au langage C

l'instruction do

Exemple 3 : lire un nombre avec contrôle de son appartenance à un intervalle de définition [Xmin,Xmax]

```
          ...
do
{  printf("valeur de x ? ");
   scanf("%f",&x);
}
while ((x < Xmin) || (x > Xmax));
          ...
```

Introduction au langage C

Les instructions conditionnelles:

instruction if

instruction switch

expression conditionnelle ?:

Introduction au langage C

Les instructions conditionnelles:

L'instruction if :

```
if ( expression )
   instruction
```

ou :

```
if ( expression )
   instruction_1
else
   instruction_2
```

Introduction au langage C

Les instructions conditionnelles:

L'instruction if :

Exemple 1 :

```
if (a<b)   mini = a;
else       mini = b;
```

Introduction au langage C

Les instructions conditionnelles:

L'imbrication des instructions if :

```
if  ( expression-1)
   instruction-1
else if ( expression-2)
   instruction-2
else if ( expression-3)
   instruction-3
...
else
   instruction-n
```

Introduction au langage C

Les instructions conditionnelles:

L'imbrication des instructions if , **exemple 2:**

```
if  (delta > 0)
  { ...traitement du cas delta > 0...
  }
else if (delta == 0)
  { ...traitement du cas delta = 0...
  }
else
  { ...traitement du cas delta < 0...
  }
```

Introduction au langage C

Exemple 3 : Pour le calcul de l'impôt sur le revenu, on veut calculer le montant des frais professionnels à déduire forfaitairement d'un célibataire.

La règle est : 10% du revenu imposable, dans la limite de 12 648 €, et avec un minimum de 376 €.

On utilisera les variables suivantes :

RI pour le revenu imposable,

FP pour les frais professionnels

et les constantes :

plafondFP = 12 648 euros,

plancherFP = 376 euros,

tauxFP = 10 % = 0,1

Introduction au langage C

```
#include <stdio.h>
main()
{ const float plafondFP = 12648.0,
              plancherFP = 376.0,
              tauxFP = 0.1;
  float   ri,fp;
  printf("Donnez le montant de votre revenu imposable : ");
  scanf("%f",&ri); /* on suppose que ri est "correcte" */
  fp = ri * tauxFP;
  if (fp < plancherFP)
    if (ri > plancherFP)
        fp = plancherFP;
    else fp = ri;
  if (fp > plafondFP) fp = plafondFP;
  printf("Les frais professionnels forfaitaires sont
                                de %8.2f Euros\n",fp);
}
```

Introduction au langage C

Remarque : Il peut y avoir plusieurs solutions à un problème donné, par exemple :

```
if (fp < plancherFP)
   if (ri > plancherFP)
        fp = plancherFP;
   else fp = ri;
if (fp > plafondFP) fp = plafondFP;
```

ou:

```
if (fp < plancherFP) && (ri > plancherFP)
            fp = plancherFP;
if (ri <= plancherFP) fp = ri;
if (fp > plafondFP) fp = plafondFP;
```

Il faut choisir en priorité celle qui semble la plus claire.

Du moins tant qu'il n'y a pas de problèmes de performances

Introduction au langage C

l'instruction switch

```
switch  ( expression )
{   case expression-constante : instructions
    case expression-constante : instructions
   ...
    default : instructions
}
```

Remarque1 : le cas default est facultatif

Remarque2 : les cas ne sont pas exclusifs

$\Rightarrow$ l'instruction break

Introduction au langage C

l'instruction switch

Exemple :

```
switch (c)
{ case 'a': case 'e':
  case 'i': case 'o':
  case 'u': printf("voyelle\n");
            break;
  case 'y': printf("semi-voyelle\n");
            break;
  default : printf("consonne\n");
}
```

Introduction au langage C

L'expression conditionnelle ? :

```
expression1 ? expression2 : expression3
```

Exemple : maximum de 2 nombres

```
max = (a > b) ? a : b;
```

est équivalent à :

```
if (a > b)
   max = a;
else
   max = b;
```

Introduction au langage C

Exercice récapitulatif : Calculer et afficher le plan d'amortissement d'un prêt. Les données à lire par le programme seront :

- la somme empruntée,
- le taux d'intérêt mensuel,
- le taux donnant les frais mensuels (assurances et frais divers) par rapport à la somme empruntée,
- le montant des mensualités souhaitées.

On considérera que la mensualité d'un mois donné est composée d'une part d'intérêts sur le mois écoulé, d'une part de frais, et d'une part d'amortissement.

Si S est la somme empruntée, les frais mensuels seront égaux à tauxfrais * S, tauxfrais étant défini comme constante (= 0,000725 par exemple) au début du programme.

Introduction au langage C

Analyse d'un tel problème:

- définir les variables impliquées
- définir les relations existant entre ces variables

- somme empruntée → **S**
- somme actuelle (restant à rembourser) → **Sact**
- numéro de la mensualité en cours → **i**

relation(s) ? avant la mensualité 1 (i = 0) : **Sact = S**

- montant des intérêts du mois en cours → **intérêt**
- montant des frais mensuels → **fraispar mois**

relation(s) ? fin du mois i, avant paiement de la mensualité i :

$$\mathbf{Sact_i = Sact_{i-1} + intérêt + fraisparmois}$$

Introduction au langage C

- montant des mensualités de remboursement

  ➡ **Rembparmois**

  relation(s) ? fin du mois i, **après** paiement de la mensualité i :

  $\mathbf{Sact_i = Sact_{i-1} + intérêt + fraisparmois - rembparmois}$

- taux des frais mensuels par rapport à la somme empruntée

  ➡ **tauxfrais**

  relation(s) ? **fraisparmois = S * tauxfrais**

- taux mensuel des intérêts ➡ **tauxmensuel**

  relation(s) ? **intérêt = Sact * tauxmensuel**

Introduction au langage C

- montant de l'amortissement du mois en cours

  ⇨ **capitalremb**

  relation(s) ?

  capitalremb = rembparmois - intérêt - fraisparmois

- montant total des remboursements ⇨ **totalremb**

  relation(s) ? **$totalremb_i = totalremb_{i-1} + rembparmois$**

Introduction au langage C

Conception d'une boucle, traitement répétitif

Le numéro de mensualité, **i varie de 0 ou 1 → ?**

➡ type de boucle : **while, for ou do**

	Itération i	
	début	fin
Sact	**$Sact_{i-1}$**	**$Sact_i$**
Totalremb	**$Totalremb_{i-1}$**	**$Totalremb_i$**

Introduction au langage C

Traitement du cas général...

```
fraisparmois = S * tauxfrais;
i = 0; Sact = S;
totalremb = 0;

do

{   i ++;

    interet = Sact * tauxmensuel;
    capitalremb = Rembparmois - interet - fraisparmois;
    Sact = Sact - capitalremb;

    totalremb = totalremb + Rembparmois;


        printf("%d  %f  %f  %f  %f  %f\n", i, Rembparmois,
                capitalremb, interet, fraisparmois, Sact);
    }
while   (Sact > 0);
```

initialisations...

$Sact_{i-1}$ $totalremb_{i-1}$

$Sact_i$ $totalremb_i$

condition...

Introduction au langage C

Avec prise en compte du calcul de la dernière mensualité

```
    do
{  i++;
   interet=Sact*tauxmensuel;
   if (Rembparmois > Sact+interet+fraisparmois)
   {  Rembparmois=Sact+interet+fraisparmois;
      capitalremb=Sact;
      Sact=0;
   }
   else
   {   capitalremb=Rembparmois-interet-fraisparmois;
    /* Sact=Sact-Rembparmois+interet+fraisparmois;  soit:*/
       Sact -= capitalremb;
   }
   totalremb += Rembparmois;
   printf("%4d %14.2f %14.2f %15.2f %8.2f %13.2f\n",
      i,Rembparmois,capitalremb,interet,fraisparmois,Sact);
} while (Sact>0);
```

Introduction au langage C

```
#include <stdlib.h>
/*Programme de calcul d'échéancier simplifié:
  A partir:
        du taux d'intérêt mensuel
        du taux des frais mensuels par rapport à la la somme empruntée
        du montant des mensualités de remboursement

        On considère que la mensualité d'un mois donné est composée
        d'une part d'intérêts sur le mois écoulé, d'une part de frais
        (assurance et divers frais) et d'une part d'amortissement.
*/
main()
{ const float tauxfrais = 0.000725;
  int i;
  float S,Sact,Rembparmois,interet,capitalremb;
  float fraisparmois,tauxmensuel,totalremb;
```

Introduction au langage C

/*REMARQUES*/
/*tauxfrais = taux des frais mensuels par rapport à la somme empruntée*/
/*i = numéro de la mensualité en cours*/
/*S = somme empruntée*/
/*Sact = somme actuelle, restant à rembourser*/
/*Rembparmois = montant des mensualités de remboursement*/
/*intérêt = montant des intérêts du mois en cours*/
/*capitalremb = montant de l'amortissement du mois en cours*/
/*fraispar mois = montant des frais mensuels = S * tauxfrais*/
/*tauxmensuel = taux mensuel des intérêts (1% sera donné par 0.01)*/
/*totalremb = montant total des remboursements*/
/*les calculs ne sont pas arrondis à 2 décimales : le programme*/
/*pourrait facilement être amélioré en effectuant ces arrondis*/

Introduction au langage C

```
printf("Somme empruntée ? ");
scanf("%f",&S);
printf("Remboursements mensuels souhaités ? ");
scanf("%f",&Rembparmois);
printf("Taux mensuel de l'emprunt (0.01 par exemple) ?");
scanf("%f",&tauxmensuel);
printf("\n");
if (Rembparmois<=S*(tauxmensuel+tauxfrais))
  printf("IMPOSSIBLE : remboursement trop faible\n");
else
{ Sact=S; i=0;fraisparmois=S*tauxfrais;
```

Introduction au langage C

```
if (Rembparmois<=S*(tauxmensuel+tauxfrais))
    printf("IMPOSSIBLE : remboursement trop faible\n");
else
{   Sact=S; i=0;fraisparmois=S*tauxfrais;
    totalremb=0;
    printf("durée    Paiement    capital remboursé   intérêt");
    printf("   frais   capital restant\n");
    do
    {   i=i+1;
        interet=Sact*tauxmensuel;
        if (Rembparmois > Sact+interet+fraisparmois)
        { Rembparmois=Sact+interet+fraisparmois;
```

Introduction au langage C

```
 if (Rembparmois > Sact+interet+fraisparmois)
{   Rembparmois=Sact+interet+fraisparmois;
    capitalremb=Sact;
    Sact=0;
}
else
{   capitalremb=Rembparmois-interet-fraisparmois;
   /* Sact=Sact-Rembparmois+interet+fraisparmois;  soit:*/
   Sact -= capitalremb;
}
totalremb += Rembparmois;
```

Introduction au langage C

```
        totalremb += Rembparmois;
        printf("%4d %14.2f %14.2f %15.2f %8.2f %13.2f\n",
                i,Rembparmois,capitalremb,interet,fraisparmois,Sact);
    } while (Sact>0);
    printf("\nle montant total des remboursements est : %14.2f\n",
            totalremb);
  }
}
```

Introduction au langage C : les fonctions

Fonctions standard : printf, scanf, abs, fabs, ...

Fonctions non standard :

- ***écrites par le programmeur,***
- ***prises dans des bibliothèques de fonctions...***

(# bibliothèque standard)

Utilité d'une fonction :

simplification de la réalisation d'un gros programme :

- **gain de clarté**
- **gain de place**
- **gain d'énergie**

Introduction au langage C : les fonctions

La bibliothèque de fonctions standard

<stdio.h> ⇨ Les entrées-sorties

<stdlib> ⇨ Les fonctions utilitaires

<math.h> ⇨ Les fonctions mathématiques

<string.h> ⇨ Le traitement des chaînes de caractères

<ctype.h> ⇨ Les tests de catégories de caractères

<time.h> ⇨ Le traitement de la date et de l'heure

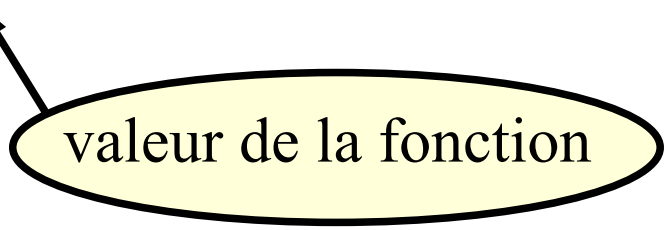
⇨ et d'autres ...

Introduction au langage C : les fonctions

<stdio.h> ⇨ Les entrées-sorties

Les premières fonctions de <stdio.h>

int printf("chaîne de format", liste d'expressions)



⇨ nombre de caractères écrits

⇨ valeur négative en cas d'erreur

Introduction au langage C : les fonctions

<stdio.h>

Les spécifications de conversion pour printf :

spécification	type de l'argument;	résultat
%d ou %i	int	--> nbre décimal
%o	int	--> nbre octal, non signé
%x ou %X	int	--> nbre hexadécimal
%u	int	--> nbre décimal non signé
%c	int	--> un caractère
%s	char *	--> chaîne de caractères
%f	double	--> [-]m.dddddd (6d par défaut)
%e ou %E	double	--> [-]m.ddddddde±xx " " [-]m.ddddddE±xx " "
%g ou %G	double	--> selon le cas %f ou %e ou %E...

Introduction au langage C : les fonctions

<stdio.h>

Modifications des spécifications :

-	cadrage à gauche
nombre	largeur minimale du résultat
.	séparateur des parties entière et décimale (nb réels)
nombre	nbre de chiffres de la partie décimale
	largeur maximale d'une chaîne de caractères

exemple1: int i=3751; printf(" ??? ",i);

???		*résultat*
x%dx	**-->**	**x3751x**
x%6dx	**-->**	**x 3751x**
x%-6dx	**-->**	**x3751 x**
x%2dx	**-->**	**x3751x**

Introduction au langage C : les fonctions

<stdio.h>

exemple2: double z = 123.456789012345;
printf(" ??? ",z);

???		résultat
x%fx	-->	x123.456789x
x%6fx	-->	x123.456789x
x%6.fx	-->	x 123x
x%6.0fx	-->	x 123x
x%10.3fx	-->	x 123.457x
x%-10.3fx	-->	x123.457 x
x%7.3fx	-->	x123.457x
x%2fx	-->	x123.456789x
x%.3fx	-->	x123.457x
x%20.9fx	-->	x 123.45678912x

Introduction au langage C : les fonctions

<stdio.h>

exemple3: char * ch = "salut à tous";

printf(" ??? ",ch);

???		résultat
x%sx	-->	xsalut à tousx
x%7sx	-->	xsalut à tousx
x%.7sx	-->	xsalut àx
x%15sx	-->	x salut à tousx
x%.15sx	-->	xsalut à tousx
x%-15sx	-->	xsalut à tous x
x%15.7sx	-->	x salut àx
x%-15.7sx	-->	xsalut à x

Introduction au langage C : les fonctions

<stdio.h>

exemple 4:

```
printf("x%3cx",'a')   -->  x  ax
printf("x%-3cx",'a')  -->  xa  x
printf("x%ux",17)     -->  x17x
printf("x%ux",-17)    -->  x4294967279x
printf("x%xx",17)     -->  x11x
printf("x%xx",-17)    -->  xfffffefx
printf("x%ox",17)     -->  x21x
printf("x%ox",-17)    -->  x37777777757x
```

Introduction au langage C : les fonctions

exemple 5: utilisation du caractère * dans un format

```
char * ch="$$$$$$$$$$$$$$$$$$$$$$$$$$$$";
int i=4,j=6;

printf("%.*s",i,ch)   -->  $$$$
printf("%.*s",j,ch)   -->  $$$$$$
```

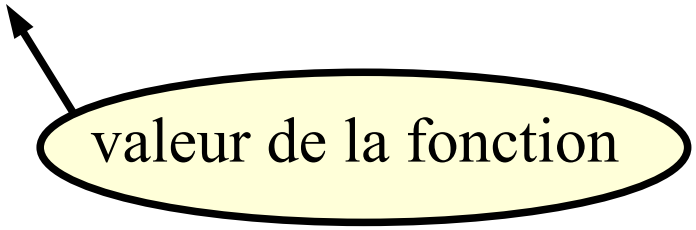
Introduction au langage C : les fonctions

<stdio.h>

<stdio.h> ➡ Les entrées-sorties

Les premières fonctions de <stdio.h>

int scanf("chaîne de format", liste d'adresses)



➡ nombre de valeurs converties et affectées

➡ EOF en cas de fichier terminé
(EOF vaut -1 et est définie dans stdio.h)

Introduction au langage C : les fonctions

<stdio.h>

Les spécifications de conversion pour scanf :

spécification	type de l'argument;	donnée lue
%d	int *	--> nbre décimal
%i	int *	--> nbre entier décimal, ou octal (précédé du chiffre 0), ou hexadécimal (0x... ou 0X...)
%o	int *	--> nbre octal, précédé ou non de 0
%x ou %X	int *	--> nbre hexa, avec ou sanx 0x ou 0X
%u	unsigned int *	--> nbre décimal non signé
%c	int *	--> un caractère
%s	char *	--> chaîne de caractères
%e, %f, %g	**float ***	--> nombre avec ou sans partie décimale, avec ou sans exposant.

Introduction au langage C : les fonctions

<stdio.h>

exemple sur scanf: (début)

```
#include <stdio.h>
main()
{ int i,j,k;
  char a,b,c;
  float x,y,z;
  double dx,dy,dz;
  char ch[30];
```

Introduction au langage C : les fonctions

<stdio.h>

exemple sur scanf: (suite 1)

```
 printf("2 entiers lus avec %%d (nnn 0nnn) ?");
scanf("%d %d",&i,&j);
printf("--> %i %i\n",i,j);
printf("3 entiers lus avec %%i (nnn 0nnn 0xnnn)?");
scanf("%i %i %i",&i,&j,&k);
printf("--> %d %d %d\n",i,j,k);
```

```
[user@localhost Desktop]$ ./a.out
2 entiers lus avec %d (nnn 0nnn) ?23 023
--> 23 23
3 entiers lus avec %i (nnn 0nnn 0xnnn)?23 023 0x23
--> 23 19 35
```

Introduction au langage C : les fonctions

<stdio.h>

exemple sur scanf: (suite 2)

```
  printf("3 float ? ");
scanf("%e %f %g",&x,&y,&z);
printf("--> %f %f %f\n",x,y,z);
printf("1 double lu avec %%f ? ");
scanf("%f",&dx);
printf("%%f--> %f     %%lf--> %lf\n",dx,dx);
printf("1 double lu avec %%lf ? ");
scanf("%lf",&dx);
printf("%%f--> %f     %%lf--> %lf\n",dx,dx);
```

```
3 float ? 1.234 2.345 3.456
--> 1.234000 2.345000 3.456000
1 double lu avec %f ? 4.567
%f--> 0.000000     %lf--> 0.000000
1 double lu avec %lf ? 4.567
%f--> 4.567000     %lf--> 4.567000
```

Introduction au langage C : les fonctions

<stdio.h>

exemple sur scanf: (suite 3)

```
  a='1';b='2';c='3';
printf("6 caractères lus avec %%c ?");
scanf(" %c%c%c",&a,&b,&c);
printf("--> %c %c %c\n",a,b,c);
scanf(" %c %c %c",&a,&b,&c);
printf("--> %c %c %c\n",a,b,c);
```

```
6 caractères lus avec %c ?xyztuv
--> x y z
--> t u v
```

Introduction au langage C : les fonctions

<stdio.h>

exemple sur scanf: (suite 4)

```
printf("1 lettre minuscule ? ");
scanf(" %c",&a);
printf("--> le caractère %c a le code %d et %x en hexa\n",a,a,a);
printf("la même lettre mais majuscule ? ");
scanf(" %c",&a);
printf("--> le caractère %c a le code %d et %x en hexa\n",a,a,a);
```

```
1 lettre minuscule ? a
--> le caractère a a le code 97 et 61 en hexa
la même lettre mais majuscule ? A
--> le caractère A a le code 65 et 41 en hexa
```

Introduction au langage C : les fonctions

<stdio.h>

exemple sur scanf: (fin)

```
    printf("chaîne de caractères ? ");
  scanf("%s",ch);
  printf("--> %s\n",ch);
  printf("chaîne de caractères ? ");
  scanf("%s",ch);
  printf("--> %s\n",ch);
}
```

```
chaîne de caractères ? le ciel est bleu...
--> le
chaîne de caractères ? --> ciel
```

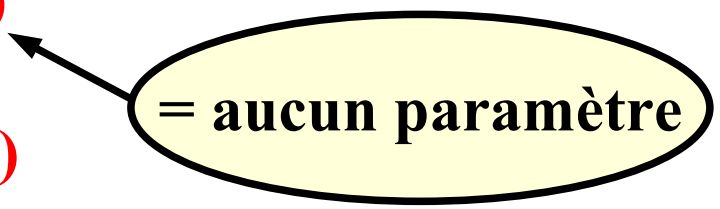
Introduction au langage C : les fonctions

<stdio.h>

2 fonctions de lecture/écriture d'1 caractère:

int getchar(void)

int putchar(int c)



***exemples*:**

```
...
char x,y='$';
...
x = getchar();
...
putchar(y);
putchar('€');
```

Introduction au langage C : les fonctions

<stdlib.h>

void exit(int status) --> **arrête le programme**
(la valeur status retournée au système)
(normalement, #0 <==> erreur)

int abs(int n) --> **|n|**

long labs(long n) --> **id. mais en long int**

Introduction au langage C : les fonctions

<math.h>

`double sin(double x)`	sinus de x
`double cos(double x)`	cosinus de x
`double tan(double x)`	tangente de x
...	
`double exp(double x)`	e^x
`double log(double x)`	ln(x), x>0
`double log10(double x)`	log10(x), x>0
`double pow(double x, double y)`	x^y, x>0 ou (x=0, y>0), ou (x<0, y entier)
`double sqrt(double x)`	racine carrée de x≥0
`double ceil(double x)`	plus petit entier ≥ x
`double floor(double x)`	plus grand entier ≤ x
`double fabs(double x)`	|x|
...	

Introduction au langage C : les fonctions

Conversion de type explicite: l'opérateur cast

(nom-de-type) expression

exemple:

```
...
int n;
double x;
...
/*on veut la racine carrée de x (de type double) */
    ....sqrt(x)...
...
/*on veut la racine carrée de n (de type entier !) */
    ....sqrt( (double) n)...
...
```

Introduction au langage C : les fonctions

<math.h>

Compilation d'un programme utilisant math.h:

gcc -lm -o myprog monprog.c --> **./myprog**

gcc -lm monprog.c --> **./a.out**

Introduction au langage C : les fonctions

Comment écrire une fonction ?

exemple 1: *valeur d'un placement **p** au taux **t** au bout de **d** années*

```
#include <stdio.h>
#include <math.h>
double val_actu(double p, double t, int d)
{ return p*pow(1.+t, (double) d);
}
main()
{ double s,taux;
  int duree;
  printf("somme placée ? ");scanf("%lf",&s);
  printf("taux annuel ? ");scanf("%lf",&taux);
  printf("durée en années ? ");scanf("%d",&duree);
  printf("--> %.2f euros après %d ans\n",
                  val_actu(s,taux,duree),duree);
}
```

paramètres formels

paramètres d'appel

Introduction au langage C : les fonctions

Règle de transmission des paramètres dans une fonction:

Les paramètres sont toujours transmis par valeur

==> un paramètre d'appel est une expression du même type que le paramètre formel correspondant.

==> il doit y avoir autant de paramètres d'appel que de paramètres formels.

Règle de transmission de la valeur d'une fonction

Si la fonction doit avoir une valeur, ceci est réalisé par l'instruction: return <expression>

où expression doit être du type de la fonction

Introduction au langage C : les fonctions

exemple 2: *Affichage d'un caractère en n exemplaires*

```
#include <stdio.h>

void aff_car(char c, int n)
{ while (n > 0) {putchar(c);n--;}
}
main()
{ int i; const int imax=5;
  for (i=1; i<=imax; i++)
  { aff_car('*',i);
    putchar('\n');
  }
}
```

Introduction au langage C : les fonctions

Trois classifications des fonctions:

1. retournant une valeur ou non.

- <type> <nom_de_fonction> ...
- void <nom_de_fonction> ... ou <nom_de_fonction> ...

2. entrées/ sorties ou non.

- fonction sans aucune lecture/écriture, ex. fonction val_actu
- fonction faisant uniquement des lectures ou écritures
- fonctions "mixtes"

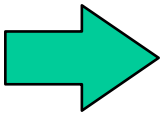
3. fonctions non récursives ou récursives.

- On verra plus tard les fonctions qui s'appellent elles-mêmes

Introduction au langage C : les tableaux

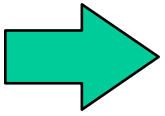
Introduction.

problème 1 : calculer la taille moyenne de n individus.



on sait faire sans tableau : une simple boucle dans laquelle on lit une taille pour l'ajouter à la somme des tailles. A la fin on divise la somme par n.

problème 2 : pour chaque individu, on veut afficher sa taille et l'écart entre sa taille et la taille moyenne.



on doit utiliser un tableau (ou un fichier) car les données doivent être traitées en 2 temps :

1. calcul de la somme et du nombre--> moyenne
2. reprise de chaque taille - moyenne --> écart

Introduction au langage C : les tableaux

introduction : exemple des écarts de taille

- Tableau des tailles.
- Constante NMAX=50 : fixe la dimension des tableaux et le nombre maximum d'individus sur les quels porte le traitement.
- Fin des données en lecture (EOF) (caractère CTRL-D)
- Variable n : compte les individus.

Introduction au langage C : les tableaux

introduction : exemple des écarts de taille

algorithme de principe :

début

- tant que la fin de fichier n'est pas rencontrée
  - lire la taille de l'individu "courant"
  - ajouter 1 au nombre d'individus
  - ajouter la taille à la somme des tailles

  (fin du tant que)
- calculer la moyenne (= somme / nombre)
- pour chaque individu faire :
  calculer et afficher sa taille et l'écart de sa taille à la moyenne

fin

Introduction au langage C : les tableaux

introduction : exemple des écarts de taille

programme "ecarts_tailles.c" : début (déclarations)

```
#include <stdio.h>
main()
{ const int NMAX=50;
  int i, n;
  float t, SommeTaille, MoyTaille;
  float tailles[NMAX];
  n=0;
  SommeTaille=0.;
```

Introduction au langage C : les tableaux

introduction : exemple des écarts de taille

programme "ecarts_tailles.c" : initialisations et 1ère boucle...

```
  n=0;
  SommeTaille=0.;
  printf("tapez les valeurs de tailles en
mètre, en finissant par CTRL-D\n");
  while ((n<NMAX) && (scanf("%f",&t)!=EOF))
  { tailles[n]=t;
    SommeTaille += t;
    n++;
  }
```

Introduction au langage C : les tableaux

introduction : exemple des écarts de taille

programme "ecarts_tailles.c" : calcul de la moyenne
+ préparation et affichage du "tableau"

```
 if (n==0) {printf("ensemble vide!!\n");exit(1);}
  MoyTaille = SommeTaille / n;
  printf("\nMoyenne des tailles : %.3f\n",MoyTaille);
  printf("\n Taille   écart à la moyenne");
  printf("\n ------   ------------------\n");
  for (i=0; i<n; i++)
     printf("%6.2f          %6.3f\n",tailles[i],
             tailles[i]-MoyTaille);
}
```

Introduction au langage C : les tableaux

introduction : exemple des écarts de taille

programme "ecarts_tailles.c" : données lues
(fichier tailles.txt)

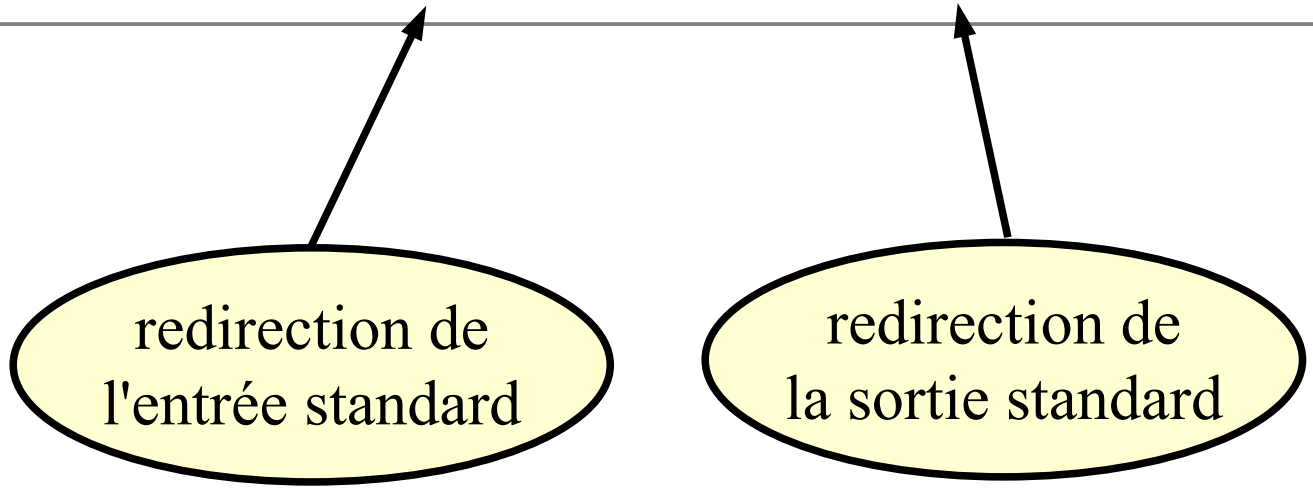
```
1.87
1.56
1.91
1.75
1.78
1.83
1.72
```

Introduction au langage C : les tableaux

introduction : exemple des écarts de taille

programme "ecarts_tailles.c" : exécution (test)
et résultat (fichier sortie.txt)

```
.....$ ./a.out <tailles.txt >sortie.txt
```



Introduction au langage C : les tableaux

introduction : exemple des écarts de taille

programme "ecarts_tailles.c" : résultat *(fichier sortie.txt)*

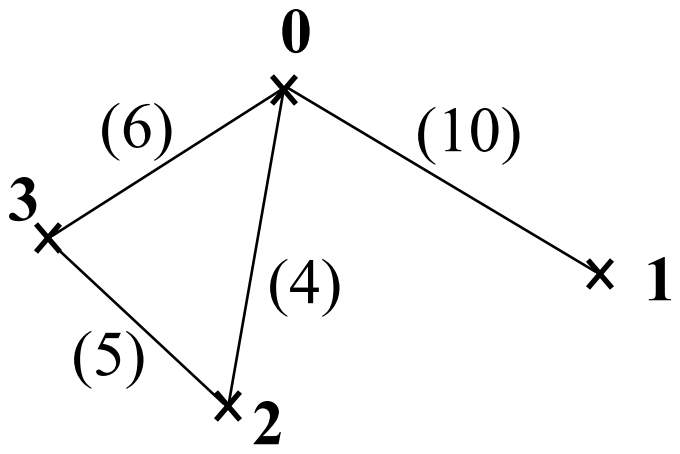
```
tapez les valeurs de tailles en mètre, en finissant par CTRL-D

Moyenne des tailles : 1.774

 Taille   écart à la moyenne
 ------   ------------------
  1.87           0.096
  1.56          -0.214
  1.91           0.136
  1.75          -0.024
  1.78           0.006
  1.83           0.056
  1.72          -0.054
```

Introduction au langage C : les tableaux

Exemple 2 : tableau à 2 dimensions

```
const int NMAX = 4;
double distances[NMAX][NMAX];
```

graphe



matrice associée du graphe

	0	*1*	*2*	*3*
0	0	10	4	6
1	10	0	-1	-1
2	4	-1	0	5
3	6	-1	5	0

Introduction au langage C : les tableaux

<référence à un élément de tableau>

1 dimension : taille [<expression entière>]

où <expression entière> ∈ [0, <dimension du tableau>]

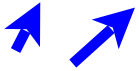
2 dimensions : tab [<expr entière1>] [expr entière2]

où <expr entière1> ∈ [0, <dimension1 du tableau>]

<expr entière2> ∈ [0, <dimension2 du tableau>]

Exemple : représentation du nombre de logements selon leur nombre de pièces, et selon le nombre d'habitants du logement.

déclaration : int nblogtcat[10] [20] of integer;



nblogtcat[3] [5] donnera le nombre de logements de 3+1 pièces où vivent 5+1 personnes.

Introduction au langage C : les tableaux

positions des éléments d'un tableau à 2 dimensions :

exemple **: int tab [2] [12];**

- **tab[0] [0], tab[0] [1], ..., tab[0] [11], tab[1] [0], ..., tab[1] [11]**

initialisation des éléments d'un tableau :

exemples **:**

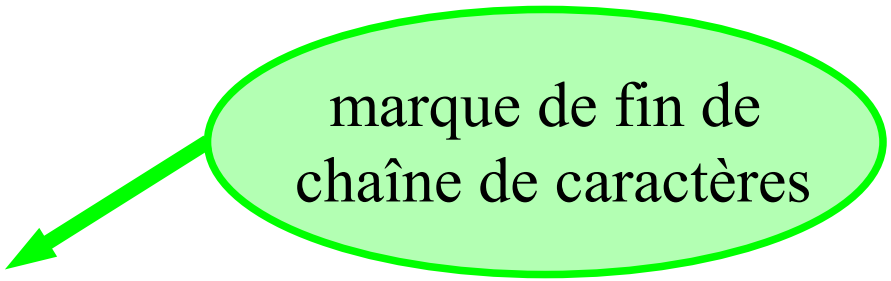
```
int tab [5] = {18, 37, 22, 31, 7};

int tab [ ] = {18, 37, 22, 31, 7};

int tab [2] [3] = {{12, 9, 11}, {33, 8, 5}};

int tab [ ] [3] = {{12, 9, 11}, {33, 8, 5}};
```

Langage C : les chaînes de caractères

***Exemple 1*: char * ch = "bonjour";**

Représentation en mémoire ?

tableau:
ch[0] : 'b'
ch[1] : 'o'
ch[2] : 'n'
ch[3] : 'j'
ch[4] : 'o'
ch[5] : 'u'
ch[6] : 'r'
ch[7] : '\0'



Langage C : les chaînes de caractères

le fichier d'entête <string.h>

int strlen(const char * cs) nombre de caractères de cs
exemple: strlen("abcd") --> 4

int strcmp(const char * cs, const char * ct) compare cs et ct:
--> valeur < 0 si cs < ct
==0 si cs == ct
> 0 si cs > ct

char * strcpy(char * s, const char * ct) copie la chaîne ct (y compris le caractère de fin '\0') dans la chaîne s.
Retourne s (= adresse de la chaîne s)

char * strncpy(char * s, const char * ct, int n) copie au plus les n premiers caractères de la chaîne ct dans la chaîne s.
Retourne s (= adresse de la chaîne s)

Remarque: pour les 2 fonctions précédentes, il est indispensable que s "pointe" sur une zone mémoire de dimension suffisante.

Langage C : les chaînes de caractères

le fichier d'entête <string.h>

Exemple:

```
char s1[]="ceci est un exemple";
char s2[11];  /* ==> toute chaîne dans s2 est limitée à 10 caractères */
char s3[100];
...
...

s2 = strcpy(s2, s1)  ==> problème !! (erreur)
...
s2 = strncpy(s2, s1, 8)  ==> c'est bon, s2 "contient" "ceci est" (+'\0')
s3 = strcpy(s3,s1) ==> c'est bon, mémoire suffisante

...
```

Langage C : les chaînes de caractères

le fichier d'entête <string.h>

char * strcat(char * s, const char * ct) étend la chaîne s avec la chaîne ct à sa suite (concaténation). Retourne s.

char * strncat(char * s, const char * ct, int n) étend la chaîne s avec au plus n caractères de la chaîne ct à sa suite. Retourne s.

Exemple :

```
char s1[ ] = "tableaux et pointeurs ";
char s2[ ] = "sont délicats...";
char s3[50];
...
s3 = strcpy(s3,s1);  /* ou bien strcpy(s3,s1); */
s3 = strcat(s3,s2);   /* ou bien strcat(s3,s2); */
   /* ou bien*/
strcat(strcpy(s3,s1),s2);
...
```

Remarque: pour les 2 fonctions précédentes, il est aussi indispensable que s "pointe" sur une zone mémoire de dimension suffisante.

langage C : fonctions et tableaux

Exemple technique 1 : lecture "rapide" d'un vecteur.

```
void lit_vecteur(double V[], int n)
{ int i;
  ... /* message d'invite éventuel */
  for(i=0; i<n; i++)scanf("%lf",&V[i]);
}
```

Utilisation de cette fonction:

```
const NMAX = 50;
double X[NMAX];
int nx;
... /* supposons que nx a été lu/initialisé à 20 */
lit_vecteur(X, nx);
...
```

X désigne l'adresse du tableau X: X équivaut ici à &X[0], adresse du 1er élément de X.

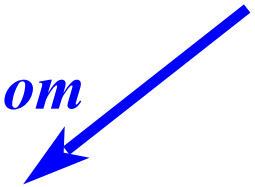
double V[] est équivalent à double * V

Les tris en mémoire

Objectif d'un tri : ordonner les éléments d'un vecteur

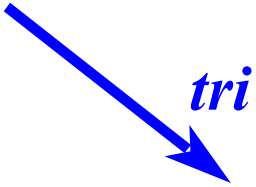
Exemple :

DUPONT	**Jean**	**30 rue du Paradis...**
ARTISTE	**Léon**	**10 allée des Peintres...**
MARTIN	**Jacques**	**162 avenue de la Plage...**



tri sur le nom

ARTISTE	Léon	10 allée ...
DUPONT	Jean	30 rue ...
MARTIN	Jacques	162 avenue ...



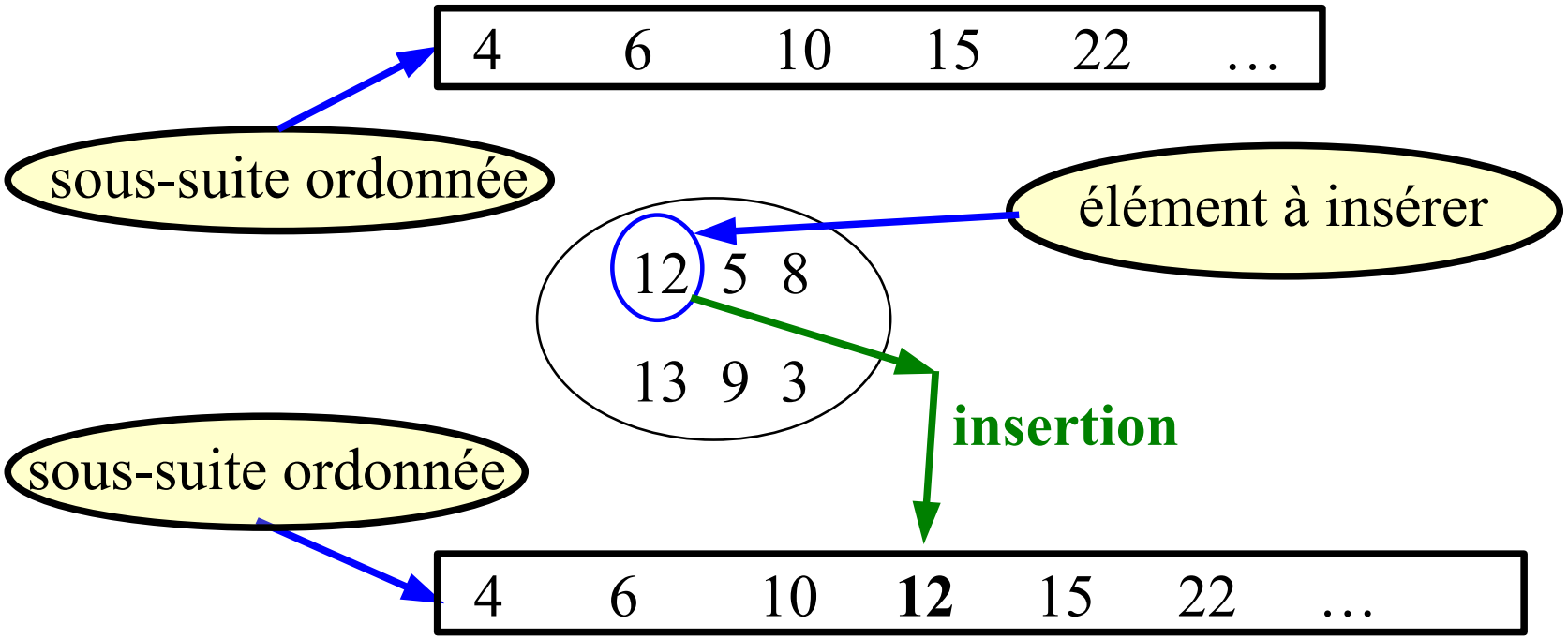
tri sur le prénom

MARTIN	**Jacques**	162 avenue ...
DUPONT	**Jean**	30 rue ...
ARTISTE	**Léon**	10 allée ...

Les tris en mémoire (suite)

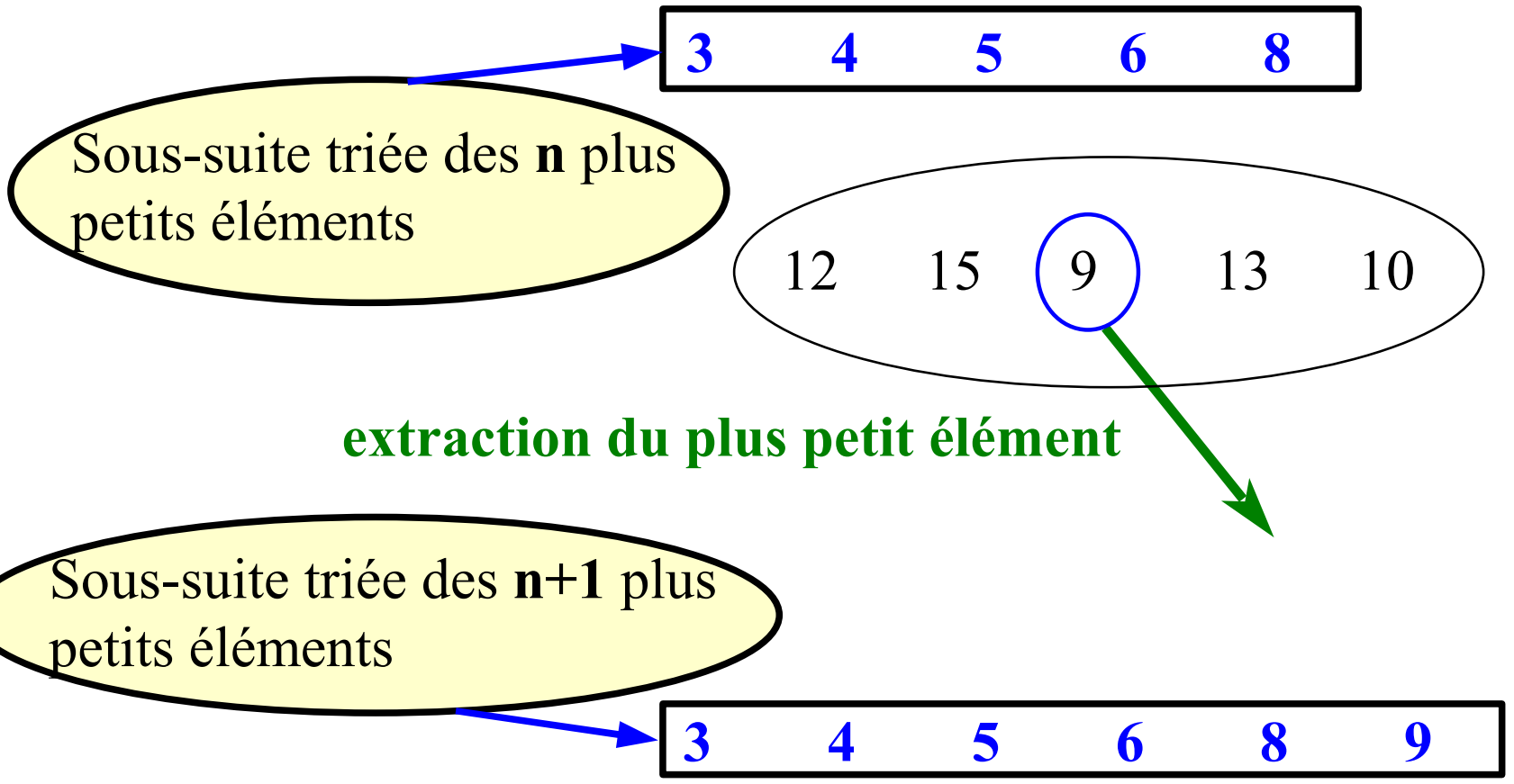
Les 3 techniques de base des tris

1. insertion :



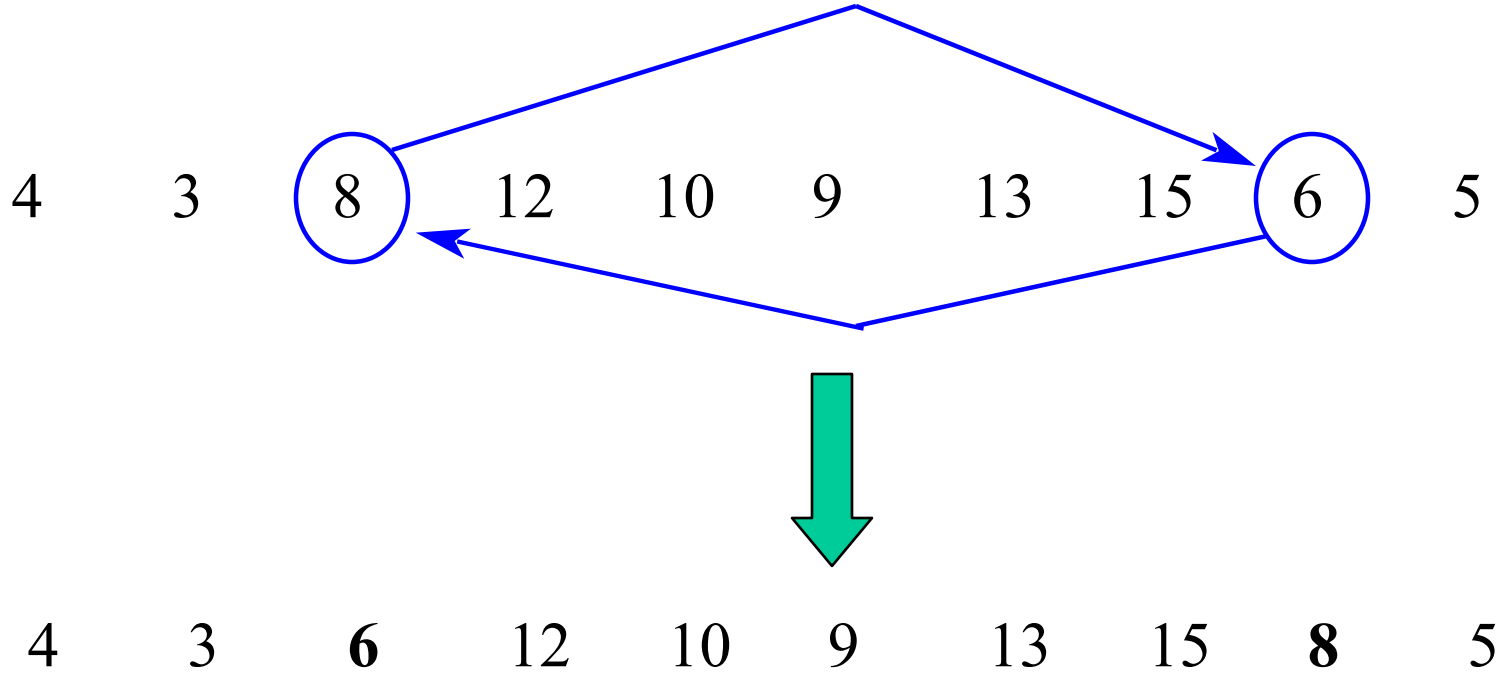
Les tris en mémoire (suite)

Extraction :



Les tris en mémoire (suite)

Échange ou permutation :



Les tris en mémoire (suite)

Le tri à bulles

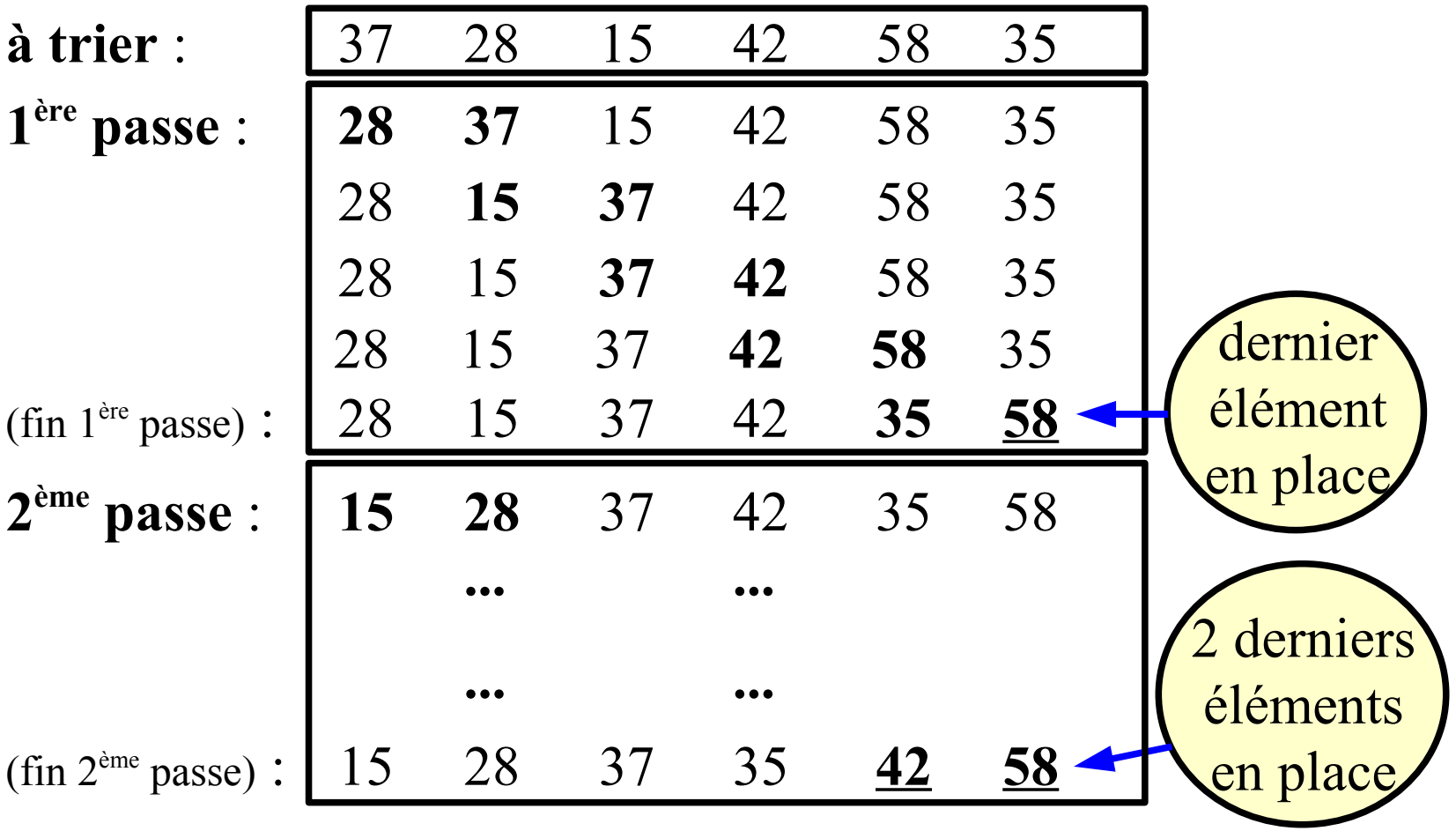
Principe du tri à bulles :

a) du début à la fin du tableau, on ordonne chaque paire d'éléments consécutifs par permutation. Ainsi, après avoir comparé, et éventuellement permuté, le ième élément et le $(i+1)^{ème}$, on compare le $(i+1)^{ème}$ et le $(i+2)^{ème}$**,...**

b) Si aucune permutation n'a été effectuée sur l'ensemble du tableau, le tri est terminé, sinon on recommence l'étape a).

Les tris en mémoire (suite)

Le tri à bulles (exemple)



Les tris en mémoire (suite)

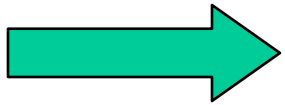
Le tri à bulles (exemple)

3ème passe (état final) :

15 28 35 **37** **42** **58**

4ème passe (état final) :

15 28 **35** **37** **42** **58**

Aucune permutation durant cette passe  **Le tri est terminé**

Les tris en mémoire (suite)

Le tri à bulles (exemple)

Fonction de permutation de 2 entiers:

```
permute (int * a, int * b)
{ int aux;
  aux = *a;
  *a = *b;
  *b = aux;
}
```

Autre façon de définir une constante (façon macro):

```
#define VRAI  1
#define FAUX  0
```

Les tris en mémoire (suite)

Le tri à bulles (exemple)

```
tri_bulle(int tab[], int n)
{ int i, j, en_ordre;
  j=n;
  do
  { en_ordre=VRAI;
    for (i=0; i<j; i++)
    { if (tab[i] > tab[i+1])
      { permute(&tab[i], &tab[i+1]);
        en_ordre = FAUX;
      }
    }
    j--;
  } while (!en_ordre);
}
```

Les tris en mémoire (suite)

Le tri par sélection ou par extraction

Principe du tri par sélection :

Début de l'itération i : **15 28 35** 42 58 37

les i-1 plus petits éléments sont déjà en place

On sélectionne/extrait le plus petit élément de la partie non triée

On permute ce plus petit élément et celui qui a la place i

15 28 35 37 58 42

Fin de l'itération i : **les i plus petits éléments sont en place.**

Les tris en mémoire (suite)

Le tri par sélection ou par extraction

Problèmes techniques à résoudre :

1. **Au début (situation initiale) aucun élément n'est en place, à la fin (situation finale) tous les éléments sont ordonnés dès qu'on a placé en ordre les n-1 plus petits éléments** (le n-ème est alors forcément à sa place).

➡ à chaque étape il faut placer le **i**ème élément (**i doit varier de 0 à n-2**)

➡ chercher le plus petit élément entre les positions i et n-1 (variable **j**).

➡ pour ce plus petit élément, il faut conserver :

	initialisations
▪ sa valeur (variable **min**)	**tab[i]**
▪ sa position (variable **jmin**)	**i**

➡ A la fin de la ième étape ?

il faut **permuter le min trouvé et le ième élément**

Les tris en mémoire (suite)

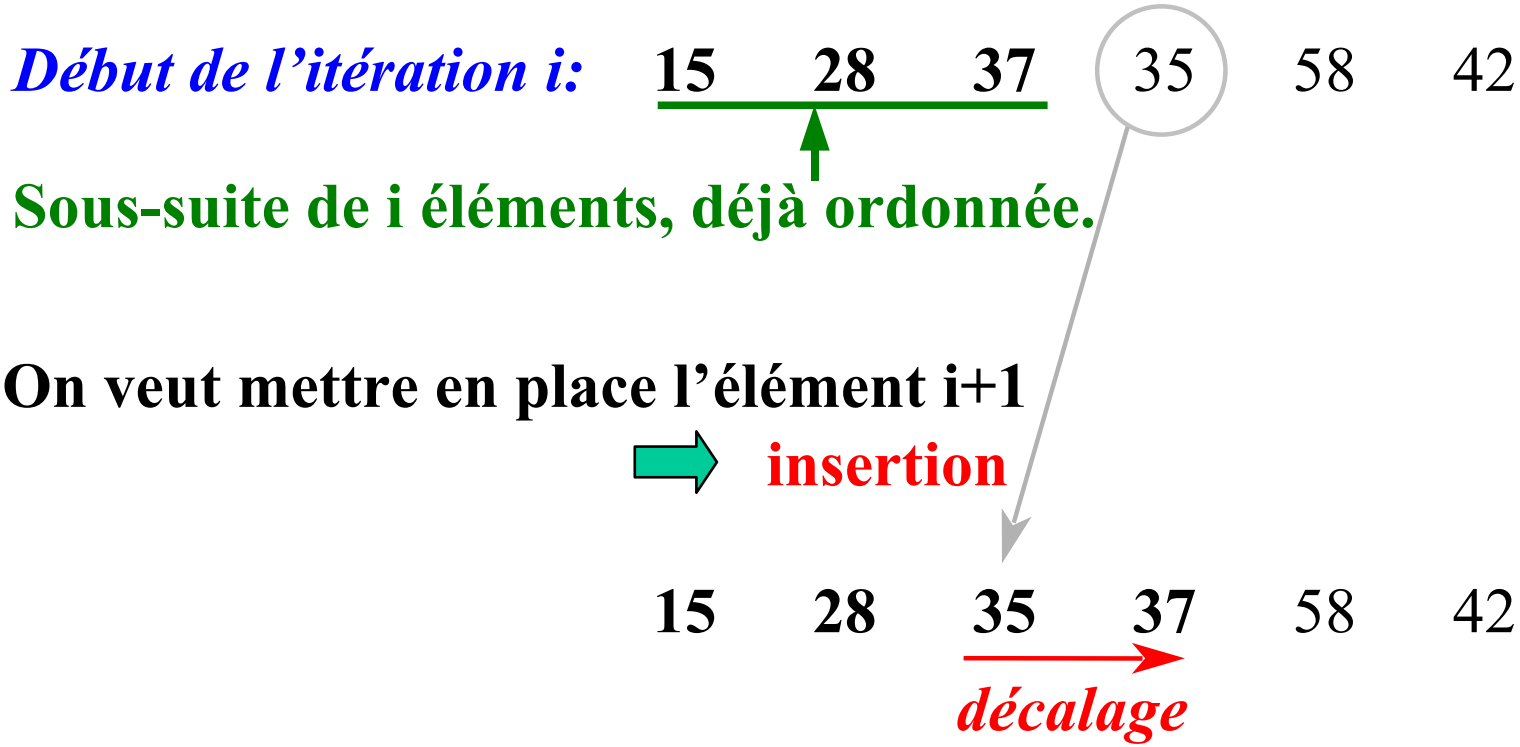
Le tri par sélection ou extraction (exemple)

```
tri_selection(int tab[], int n)
{ int i, j, min, jmin;
  for (i=0; i<n-1; i++)
  { min=tab[i]; jmin=i;
    for (j=i+1; j<n; j++)
      if (tab[j]<min)
      { min=tab[j];
        jmin=j;
      }
    tab[jmin]=tab[i];
    tab[i]=min;
  }
}
```

Les tris en mémoire (suite)

Le tri par insertion

Principe du tri par insertion :



Les tris en mémoire (suite)

Le tri par insertion

Problèmes techniques à résoudre :

1. l'insertion : c'est la difficulté essentielle.

A priori, il y a 3 cas distincts, selon la position d'insertion du nouvel élément :

- au début de la sous-suite ordonnée,
- à la fin de la sous-suite ordonnée,
- entre ces 2 positions extrêmes.

Les tris en mémoire (suite)

Le tri par insertion

Exemple d'insertion incorrecte (= 1er essai) :

```
/* on suppose que tab[i] peut être écrasé */
    aux=tab[i]; j=i-1;
    while (tab[j]>aux)
    { tab[j+1]=tab[j];
      j--;
    }
    tab[j+1]=aux;
```

On constate que si aux < tab[0], on est amené à comparer tab[-1] et aux

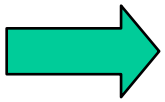
pb car tab[-1] n'est pas concerné et n'existe pas !

Les tris en mémoire (suite)

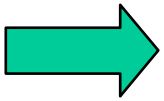
Le tri par insertion

Problème de l'insertion (suite) :

Risque d'erreur en référençant tab[-1]



il ne faut pas référencer tab[-1].



il faut tester la valeur de i
avant le test tab[j]>aux

... il existe une autre solution:
voir plus loin

Les tris en mémoire (suite)

Le tri par insertion (exemple)

```
tri_insertion(int tab[], int n)
{ int i, j, aux;
  for (i=1; i<n; i++)
  { aux=tab[i]; j=i-1;
    while ( (j>=0) && (tab[j]>aux) )
    { tab[j+1]=tab[j];
      j--;
    }
    tab[j+1]=aux;
  }
}
```

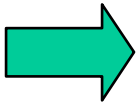
Les tris en mémoire (suite)

Le tri par insertion

Performance des algorithmes de tri (Complexité en temps):

On veut avoir une idée de la durée d'un algorithme, compte tenu de la dimension des données traitées.

Pour les tris, pour n éléments, on veut savoir de quelle façon la durée d'exécution d'un programme dépend de n.



on va évaluer le nombre d'instructions exécutées, au maximum (on dit **au pire cas**)

Tri à bulles : $5\ n^2 / 2$

Tri par sélection : $3\ n^2 / 2$

Tri par insertion : $2\ n^2$

Les tris en mémoire (suite)

Le tri par insertion

Notion de sentinelle.

Une sentinelle est un élément, éventuellement ajouté artificiellement, placé en limite d'un espace de recherche, et dont la valeur est telle que la recherche va s'arrêter au pire sur cet élément.

Amélioration du tri par insertion.

Dans le tri par insertion, on va faire en sorte que le plus petit élément de la suite à trier soit placé en tête, à sa place définitive, où il fonctionnera en sentinelle. Cette opération préliminaire a une durée proportionnelle à n et est donc négligeable par rapport à la durée du tri qui reste de l'ordre de n^2.

Avec cette amélioration la complexité passe de $2.n^2$ à $c.n^2$.

($c<2$, on divise le nombre de tests par 2)

Les tris en mémoire (suite)

Le tri par insertion avec sentinelle

Recherche du plus petit élément pour le placer en sentinelle

```
min=tab[0]; imin=0;
for (i=1; i<n; i++)
   if (tab[i]<min){ min=tab[i]; imin=i;}
tab[imin]=tab[0];
tab[0]=min;
```

Les tris en mémoire (suite)

Le tri par insertion avec sentinelle

Sentinelle ⇨ pas de risque d'adressage de tab[-1], car **tab[0] fait office de sentinelle**.

⇨ tab[0] et tab[1] sont en ordre,

⇨ on commence avec i = 2

```
for (i=2; i<n; i++)
  { aux=tab[i]; j=i-1;
    while (tab[j]>aux)
    { tab[j+1]=tab[j];
      j--;
    }
    tab[j+1]=aux;
  }
```

Les tris en mémoire (suite)

Conclusion sur les tris

Examen du choix de ces 3 méthodes de tri:

- **elles sont parmi les moins rapides.**

\+ **elles sont simples.**

++ **elles regroupent quelques techniques de base de l'algorithmique:**

- **insertion d'un nouvel élément dans une liste ordonnée,**
- **recherche du plus petit élément d'une liste,**
- **permutation de 2 éléments,**
- **utilisation d'une sentinelle.**

Les tris en mémoire (suite)

Conclusion sur les tris

Les meilleurs algorithmes de tri ont une complexité (en temps) de l'ordre de n.log(n).

Les tris de fichiers

un fichier ne peut pas tenir entièrement en mémoire,
temps d'accès mémoire et disque : de 1 à 1000,
performences ≡ réduction du nombre d'accès disque,
==> utilisent des méthodes très différentes.

Fonctions récursives

Principe : une fonction peut s'appeler elle-même. Ainsi la définition d'une fonction, peut être très semblable à sa définition par récurrence.

Exemple 1 : définition de la fonction factorielle par récurrence.

On définit $u_n = n!$ de la façon suivante (par récurrence):

$$u_0 = 1$$
$$\forall n > 1,\ u_n = n.u_{n-1}$$

Fonctions récursives

Exemple 1 de fonction récursive

```
int factnrec( int v)
{ if (v == 0)
      return 1;
      else return v * factnrec(v-1);
end;
```

Au moins un cas «non récursif»

arrêt possible

Fonction référencée dans une expression

= appel récursif.

Fonctions récursives

Utilisation de la récursivité à bon escient :

exemple 1 : calcul de n!. **(+/-)**

acceptable mais moins bonne que la solution itérative toute simple basée sur une boucle for (...;for(aux=i=1;i<=n;i++) aux*=i;...), qui est en fait plus efficace.

exemple 2 : les tours de Hanoi. **(+ +)**

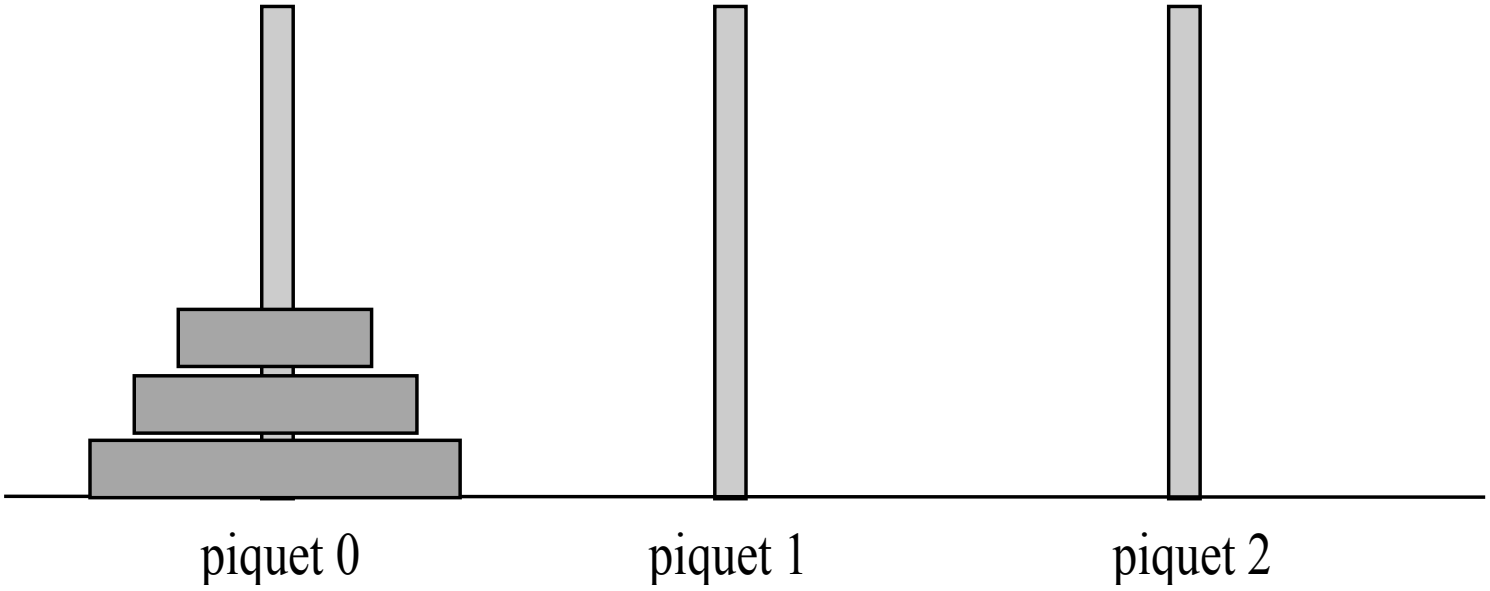
beaucoup plus simple qu'une solution itérative. Dans la même catégorie (algorithmes avantagés par la récursivité), parmi les algorithmes de tris les plus efficaces, certains sont naturellement récursifs.

exemple 3 : suite de Fibonacci. **(- - -)**

la solution récursive peut être catastrophique. C'est le cas de la (mauvaise) solution récursive qui sera donnée.

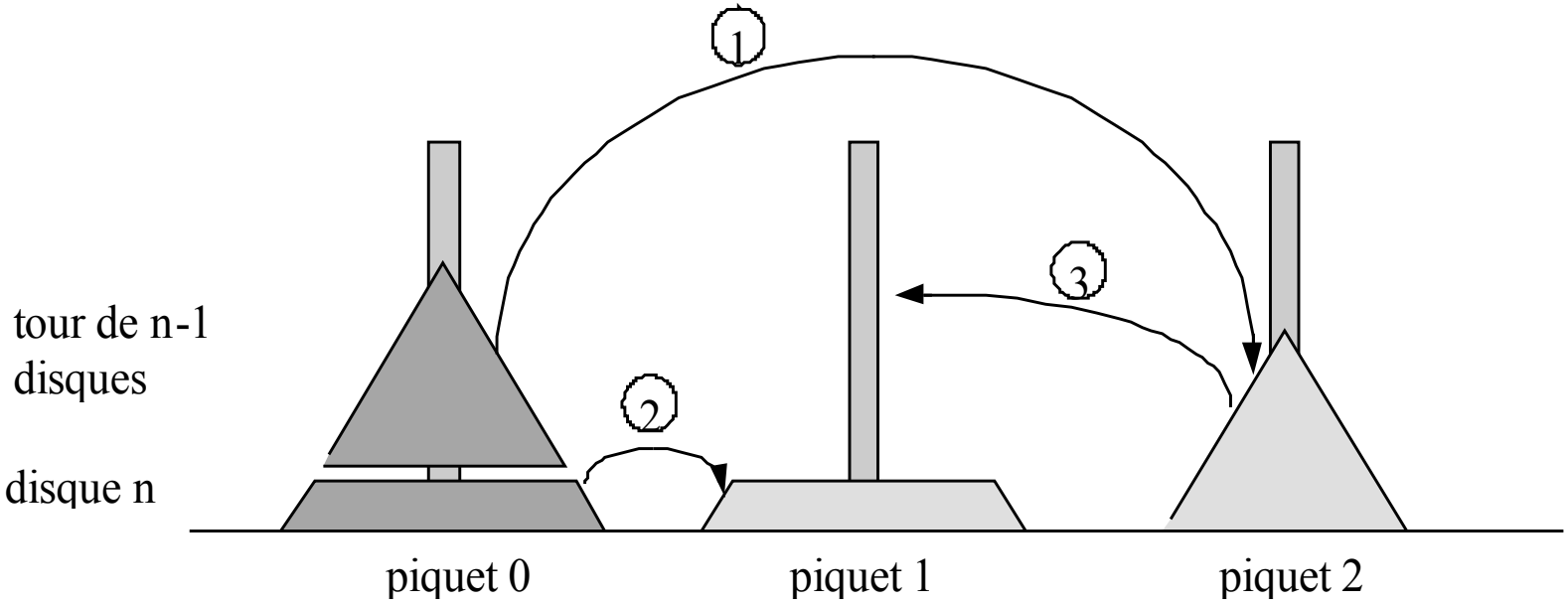
Fonctions récursives: tours de Hanoï

Récursivité, exemple 2 : les tours de Hanoï.



Fonctions récursives: tours de Hanoï

principe de l'algorithme récursif *(exemple 2)*



1: appel récursif pour n-1 disques, du piquet 0 (pqd) vers le 2 (pqt)

2: afficher le déplacement du disque n du piquet 0 (pqd) vers le 1(pqa)

3: appel récursif pour n-1 disques, du piquet 2 (pqt) vers le 1 (pqa)

Fonctions récursives: tours de Hanoï

```
#include <stdio.h>
#define NMAX 4  /*nombre maximum de disques de la tour a deplacer
*/

void deplacer_tour(int n, int pqd, int pqa, int pqt)
{ if (n == 1)
    printf("   disque 1: piquet %d --> piquet %d\n",pqd,pqa);
    else
    { deplacer_tour(n-1,pqd,pqt,pqa);
      printf("   disque %d: piquet %d --> piquet %d\n",n,pqd,pqa);
      deplacer_tour(n-1,pqt,pqa,pqd);
    }
}

main()
{ int nd; /* nbre effectif de disques de la tour à deplacer */
  printf("\n\n%30s\n","Tours de Hanoi");
  printf("%30s\n\n","==============");
  printf("Combien de disques voulez-vous (de 1 a %d ? ",NMAX);
  scanf("%d",&nd);
  deplacer_tour(nd,0,1,2);
}
```

Fonctions récursives: tours de Hanoï

Amélioration de la procédure deplacer_tour.

Condition de fin des appels récursifs ?

Que se passe-t-il avec n > 0 au lieu de n > 1 ?

- le cas n=0 est alors traité non récursivement..., en ne faisant rien!
- le cas n = 1 est traité récursivement ... correctement

```
void deplacer_tour(int n, int pqd, int pqa, int pqt)
{ if (n > 0)
    { deplacer_tour(n-1,pqd,pqt,pqa);
      printf("   disque %d: piquet %d --> piquet %d\n",n,pqd,pqa);
      deplacer_tour(n-1,pqt,pqa,pqd);
    }
}
```

Fonctions récursives: tours de Hanoï

Autre amélioration de la procédure deplacer_tour.

On a la relation suivante :

$$pqd + pqa + pqt = 0+1+2 = 3$$

➡ ***on peut «gagner» un paramètre, le dernier par exemple (pqt).***

```
void deplacer_tour(int n, int pqd, int pqa)
{ if (n > 0)
    { deplacer_tour(n-1,pqd,3-pqa-pqd);
      printf("   disque %d: piquet %d --> piquet %d\n",n,pqd,pqa);
      deplacer_tour(n-1,3-pqa-pqd,pqa);
    }
}
```

Amélioration faible, perte de clarté.

Récursivité, exemple 3 : suite de Fibonacci.

$$\begin{cases} F_0 = 0 \\ F_1 = 1 \\ \forall i > 1,\ F_i = F_{i-1} + F_{i-2} \end{cases}$$

1ère solution: non récursive (la meilleure) :

```
#include <stdio.h>
int fibo_ite(int n)
{ int i,fn,fnm1,fnm2;
  fnm1=1; fnm2=0;
  if (n==1) fn=1; else fn=0; /* au cas où n<=0 */
  for (i=1;i<=n;i++)
  { fn=fnm1+fnm2;
    fnm2=fnm1;
    fnm1=fn;
  }
  return fn;
}
main()
{ int n;
  printf("indice du terme désiré de la suite de Fibonacci ? ");
  scanf("%d",&n);
  printf("Le terme d'indice %d vaut: %d\n",n,fibo_ite(n));
}
```

Récursivité, exemple 3 : suite de Fibonacci.

Limites de la solution itérative :

F_{46} (=1.836.311.903) est le plus grand terme «tenant» dans le type int utilisé ici.

Complexité de la solution itérative :

Elle est de l'ordre de n (structure limitée à une seule boucle for de 1 à n).

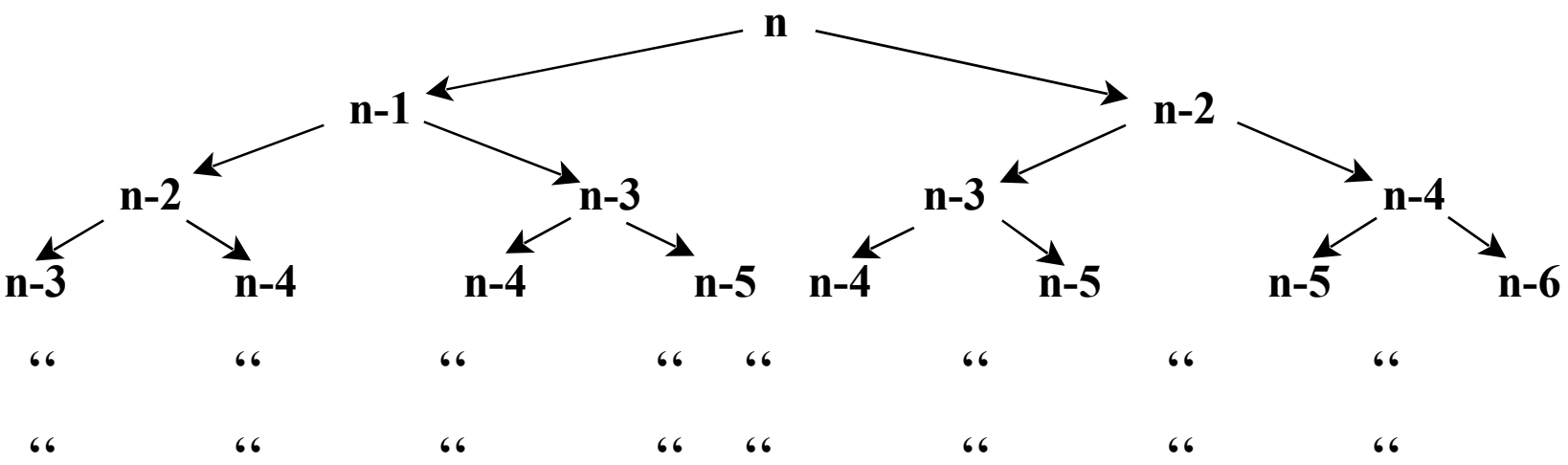
Fonction récursive fib_rec(n) *(solution très maladoite)*:

```
int fibo_rec(int n)
{ if (n<=0) return 0;
  else if (n==1) return 1;
  else return fibo_rec(n-1)+fibo_rec(n-2);
}
```

Récursivité, exemple 3 : suite de Fibonacci.

Complexité de cette solution récursive :

arborescence des appels récursifs:

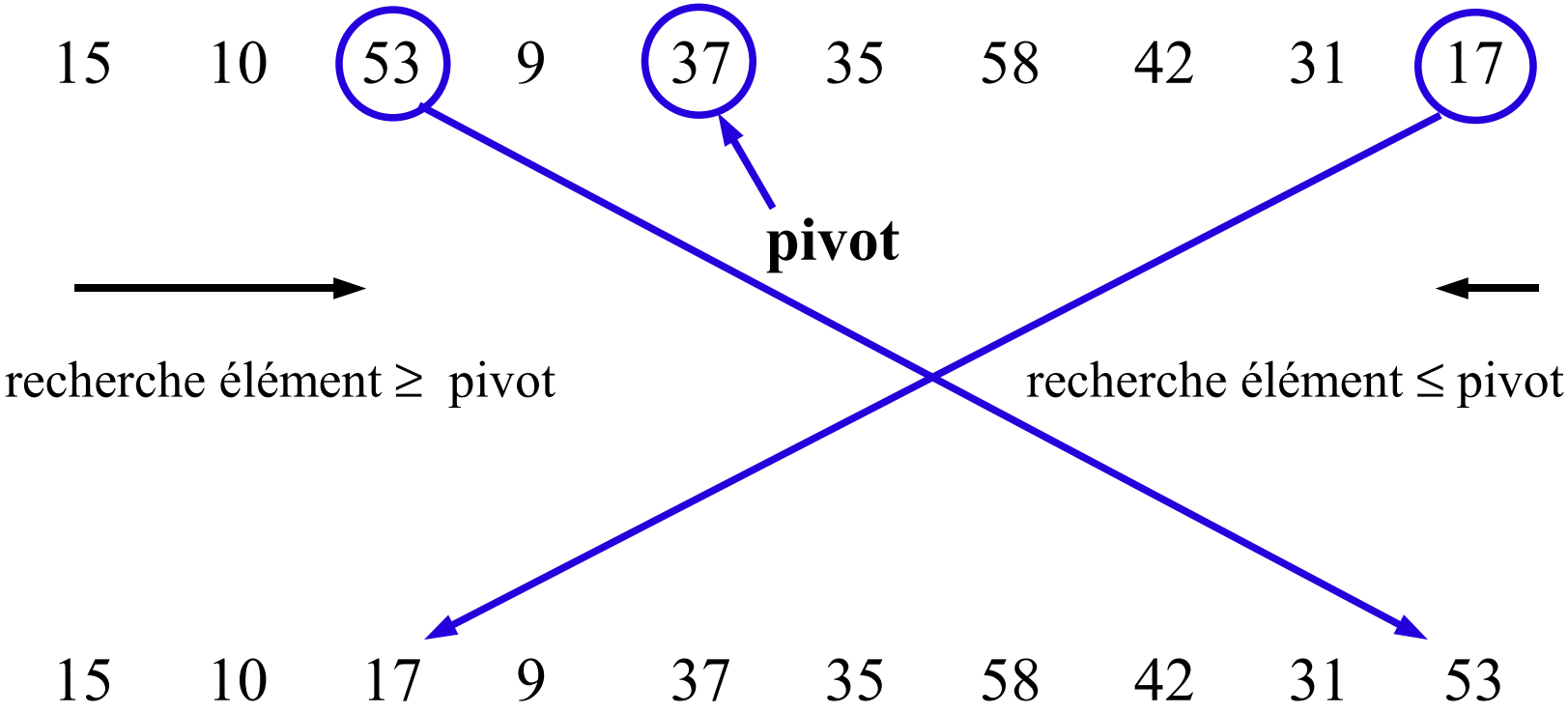


Pour n >= 2 chaque appel génère 2 appels récursifs.

au total on aura de l'ordre de 2^n appels récursifs

Récursivité, exemple 4 : tri rapide (quicksort)

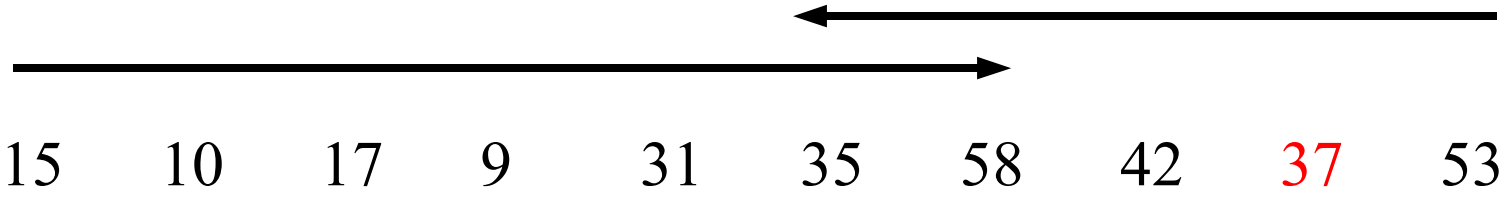
Principe du tri rapide :



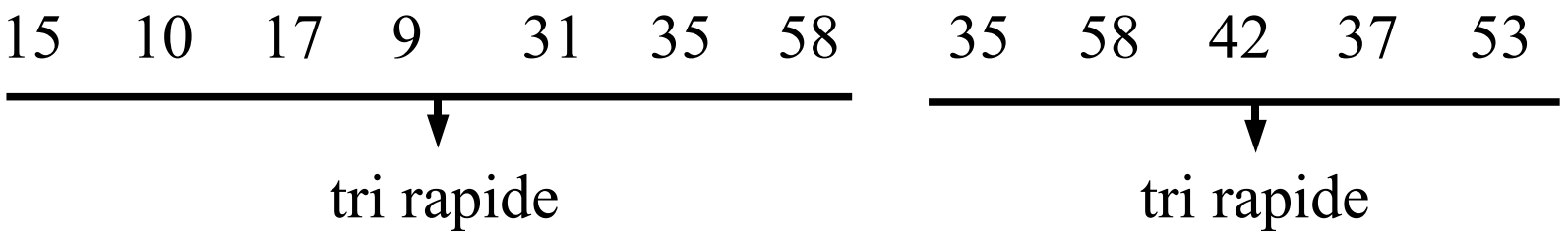
Récursivité, exemple 4 : tri rapide (quicksort)

Principe du tri rapide :

1ère partie: séparation autour du pivot



2ème partie: tri (récursif) des 2 parties si nécessaire



Récursivité, exemple 4 : tri rapide (quicksort)

```
quick_sort(int T[],int d,int f)
{ int p, vp, i, j,aux;
  vp=T[(d+f)/2]; /* vp=pivot */
  i=d; j=f;
  do
  { while (T[i]<vp) i++;
    while (T[j]>vp)j--;
    if (i<=j)
    { aux=T[i]; T[i]=T[j]; T[j]=aux;
      i++;j--;
    }
  } while (i<=j); /* fin partie "séparation" */
  if (d<j) quick_sort(T,d,j);
  if (i<f) quick_sort(T,i,f);
}
```

durée moyenne en n.logn

Exercices de révision

EC1 : écrire un programme qui calcule le nombre de phrases et le nombre moyen de caractères par phrase d'un texte donné.

Question 1 : Comment reconnaître une phrase ?

Question 1bis : Y-a-t-il des cas plus difficiles ?
A ignorer dans un premier temps...

Question 2 : Quels caractères compter ?
... ou ne pas compter ?

Question 2bis : peut-on simplifier dans un 1er temps ?
... puis compléter facilement ?

Exercices de révision

EC2 : Ecrire un programme qui lit un nombre entier et affiche aussi proprement que possible sa décomposition en facteurs premiers.

2 questions initiales :

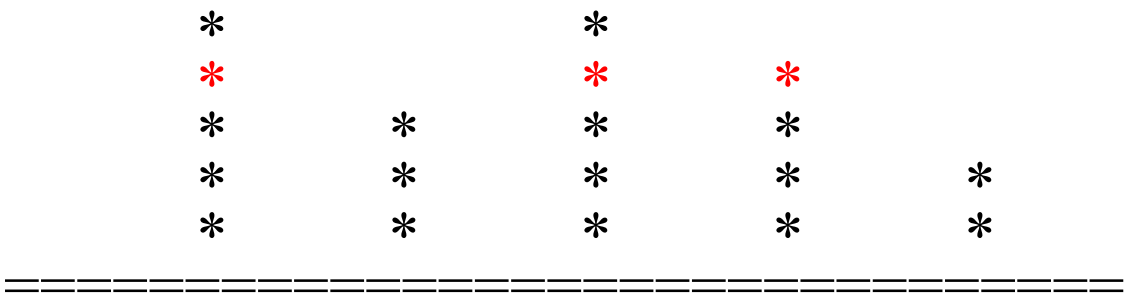
1. **Algorithme de recherche des facteurs premiers**
2. **Affichage de la décomposition**

2. trois exemples caractéristiques:

pour 11 : « ==> nombre premier »
pour 616 : « 2^2 * 7 * 11 »
pour 225 : « 3^2 * 5^2 »

Exercices de révision

EC3 : Ecrire une procédure qui retranscrit un tableau de valeurs entières en un diagramme en bâtons verticaux.



idée: l'affichage se fait ligne par ligne:

Présence d'un bâton à un niveau donné --> *

Absence d'un bâton (=on est au-dessus) --> espace

Exercices de révision

EC4 : Ecrire un programme qui réalise une donne aléatoire d'un jeu de 52 cartes à 4 joueurs.

***Aides*:**

- **si x est un entier, la fonction random(x) donne un nombre aléatoire entier a ∈ [0,x[.**
- **La procédure randomize, utilisée sans paramètre, permet d'initialiser le générateur de nombres (pseudo)aléatoires différemment (c'est l'horloge système qui est utilisée).**
- **Attention : une carte donnée n'est présente qu'une fois dans un jeu de cartes !**

Exercices de révision

EC5 : Ecrire un programme faisant jouer l'ordinateur contre l'utilisateur au jeu suivant:
On dispose au départ de 50 allumettes. Chaque joueur à tour de rôle retire des allumettes du tas, au moins 1 et au plus 6. Celui qui prend la dernière perd.

(Cet exercice est tiré de "Jeux et casse-tête à programmer" de Jacques Arsac aux éditions DUNOD (1985)).