

# Les Bases du C

# Un peu de culture ...

- 1978 : version K&R
  - Créée par le duo Brian W. Kernighan / Dennis M.
  - Née pour développer une version portable du système d'exploitation UNIX
- 1989 : version **ANSI-C**
  - Créée par l'organisme de normalisation américain.
  - Née pour normaliser le C, car les éditeurs de compilateur ont créé des variantes
- 1990 : ANSI C ++

# Caractéristiques du C

- Procédural
- Fortement typé
- Compact : basé sur un noyau de fonctions et d'opérateurs limité.
- Portable : un même programme pour plusieurs systèmes d'exploitation simplement en le recompilant, à condition de respecter la norme ANSI.
- Extensible : bibliothèques supplémentaires de fonctions .

# Comment pratiquer le C

- Soit avec un simple éditeur de texte (Ex: notepad ++) et un compilateur ( Gcc pour linux, mingw pour Windows )

**J'écris mon programme**



```
1  #include <stdio.h>
2  int main ( )
3  {
4      // affiche bonjour !!
5      printf("Bonjour !! ");
```

**J'ouvre une fenêtre de commande et lance la compilation.**

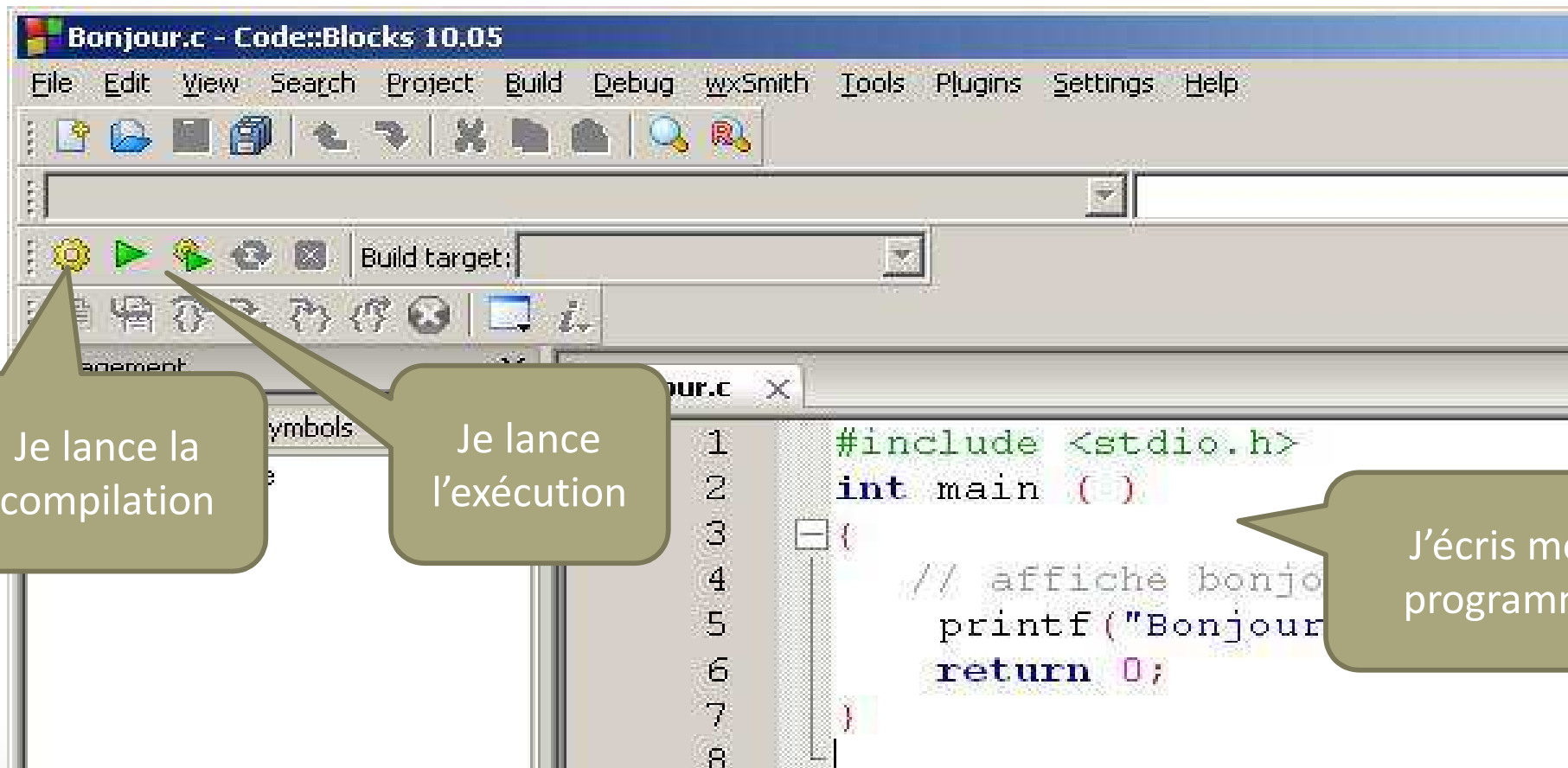
```
gcc bonjour.c -o bonjour.exe
```

**Puis je lance l'exécution**

```
./bonjour.exe
```

# Comment pratiquer le C

- Soit avec un environnement intégré. Ex : codeBlocks



# Structure d'un programme C

Le compilateur a besoin d'une fonction principale, appelée « main » : c'est le point d'entrée d'un programme.

DEBUT

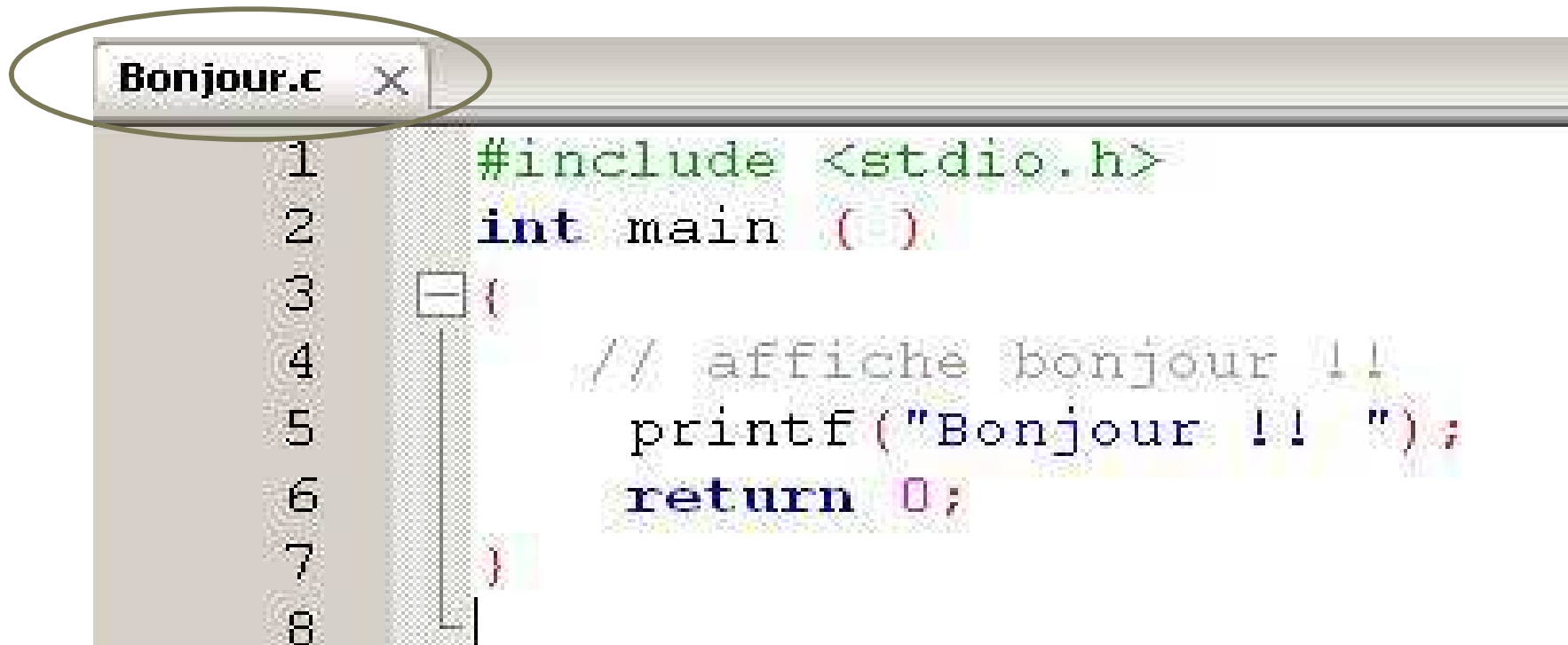
```
int main ()  
{  
    /* commentaires  
       sur plusieurs lignes */  
    instruction ;  
    instruction ;  
    // commentaire sur une ligne ...  
    return 0 ;  
}
```

FIN

retourne code erreur  
0 ⇔ pas de pb !

# Structure d'un programme C

Le nom du fichier indique le nom du programme.



```
1  #include <stdio.h>
2  int main ( )
3  {
4      // affiche bonjour !!
5      printf("Bonjour !! ");
6      return 0;
7  }
```

The image shows a code editor window with the title bar "Bonjour.c" circled in green. The code inside is a C program that includes the standard input/output library, defines a main function, and prints "Bonjour !!". The code is color-coded: keywords like #include, int, printf, return, and the closing brace are in blue; the string "Bonjour !! " is in red; and the rest is in black. Line numbers 1 through 8 are visible on the left side of the editor.

# Variables en C

- **Les variables doivent être déclarées et typées.**
- **Il existe 4 types primitifs : ce sont 4 types numériques.**
  - Pas de type booléen
  - Les caractères sont avant tout des numériques d'après la table ASCII
    - A-Z : 65-90
    - a-z : 97-122
  - Les chaînes sont des tableaux de caractères => ensemble de numériques



# Les types entiers en C

Type	Taille (octet)	Valeur minimum	Valeur maximum
char	1	-128	127
short	2	-32 768	32 767
long	4	-2 147 483 648	2 147 483 647
int	4	-2 147 483 648	2 147 483 647

unsigned char	1	0	255
unsigned short	2	0	65 535
unsigned int	4	0	4 294 967 295
unsigned long	4	0	4 294 967 295

# Les types réels en C

Type	Mantisse	Taille (octet)	Valeur minimum	Valeur maximum
float	6	4	$3.4 * 10^{-38}$	$3.4 * 10^{38}$
double	15	8	$1.7 * 10^{-308}$	$1.7 * 10^{+308}$
long double	19	10	$3.4 * 10^{-4932}$	$3.4 * 10^{+4932}$

# Déclarer une variable

Il est préférable de déclarer les variables en début de programme.

```
int main  
{  
    float longueur ;  
    float largeur, perimetre;
```

# Déclarer une constante

Il faut initialiser la constante dès sa déclaration.

```
int main  
{  
    float const COEFF= 1.852 ;
```

# Affecter une variable

Grâce à une expression.

```
int main
{
    float longueur ;
    float largeur, perimetre;
    longueur = 5 ;
    largeur = 2 ;
    perimetre = 2 * (longueur + largeur);
}
```

# Opérations

- Addition, soustraction, multiplication : + - \*
- Incrémentation, décrémentation : ++ --

```
int res = 2 ;
```

```
res ++ ;
```

⇔ res = res + 1 ; => 3

```
res --;
```

⇔ Res =res -1 ; => 2

- Division: /

Attention : la division d'entier donne un résultat entier.

```
float res = 5/2 ;
```

2

- Modulo : % (reste de la division entière)

Attention : la division d'entier donne un résultat entier.

```
int reste = 5%2 ;
```

1

# Opérations

La priorité des opérations définie en mathématique est respectée.

```
int largeur = 3 ;
```

```
int longueur = 3+ largeur* 2;
```

9

```
int largeur = 3 ;
```

```
int longueur = (3+ largeur)* 2;
```

12

# Affecter une variable : scanf

Grâce à une saisie utilisateur => fonction **scanf**.

**Prototype : int scanf(const char \* format, ...);**

Elle attend 2 paramètres au minimum :

- Format de la donnée attendue
- L'adresse mémoire de la variable à affecter

**scanf( "%i", &largeur ) ;**



Parametre 1 : format  
de la valeur saisie  
par l'utilisateur : ici  
type int



Parametre 2 : adresse  
mémoire de la variable qui  
va stocker la valeur saisie :  
ici largeur



# Affecter une variable : scanf

Les formats les plus courants :

Type	format
int, short	%d ou %i
long	%ld
float	%f
double	%lf
char	%c

```
printf ("Tapez F pour Femme, H pour Homme\n");  
scanf("%c", &genre);  
printf ("Entrez votre age\n");  
scanf("%i", &age);  
printf (" Entrez votre taille\n");  
scanf("%f", &taille);
```

# Affecter une variable : scanf

Le format doit être compatible avec le type déclaré !

**int** age ;  
....  
scanf( "%i" , &age );

A diagram illustrating the compatibility between a variable type and a scanf format specifier. The variable type 'int' is circled in the declaration 'int age ;'. The format specifier '%i' is circled in the scanf call 'scanf( "%i" , &age );'. A double-headed arrow connects the 'int' and '%i', indicating they are compatible.

**char** genre ;  
....  
scanf( "%c" , &genre );

A diagram illustrating the compatibility between a variable type and a scanf format specifier. The variable type 'char' is circled in the declaration 'char genre ;'. The format specifier '%c' is circled in the scanf call 'scanf( "%c" , &genre );'. A double-headed arrow connects the 'char' and '%c', indicating they are compatible.

**float** taille;  
....  
scanf( "%f" , &taille);

A diagram illustrating the compatibility between a variable type and a scanf format specifier. The variable type 'float' is circled in the declaration 'float taille;'. The format specifier '%f' is circled in the scanf call 'scanf( "%f" , &taille);'. A double-headed arrow connects the 'float' and '%f', indicating they are compatible.

# Affecter une variable : scanf

Le prototype de Scanf est déclarée au sein du fichier stdio.h

#include <stdio.h>

```
int main ()
{
    int age ;
    char genre ;
    float taille ;
    printf ("Tapez F pour Femme, H pour Homme\n");
    scanf("%c", &genre);
    printf ("Entrez votre age\n");
    scanf("%i", &age);
    printf (" Entrez votre taille\n");
    scanf("%f", &taille);
```

# Affecter des variables : scanf

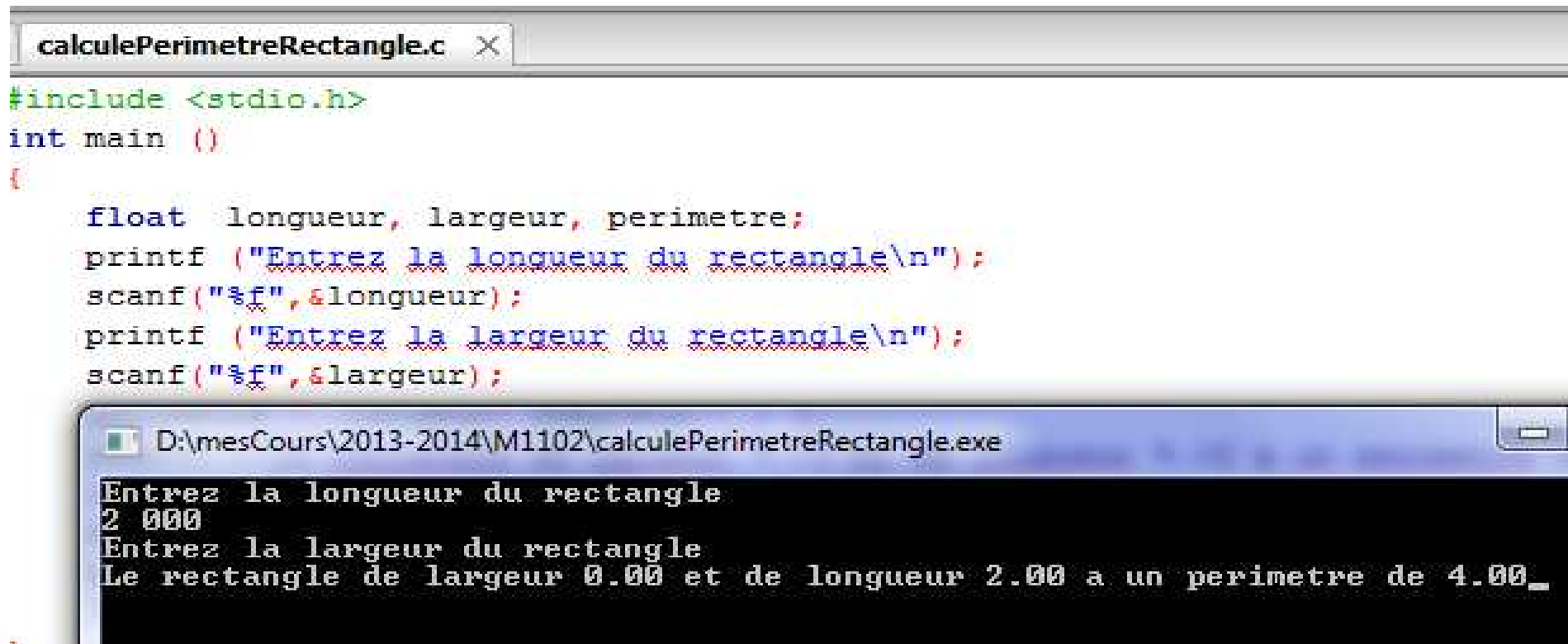
On peut affecter plusieurs variables. Mais attention, le séparateur saisi doit être identique au séparateur défini par le format.

```
#include <stdio.h>
int main ()
{
    int jour,mois, annee;
    printf ("Entrez votre date de naissance (J/M/A)\n");
    scanf("%i/%i/%i", &jour, &mois, &annee);
```

# Limites du scanf

scanf s'arrête sur un espace, une tabulation ou une entrée.

Les caractères suivants sont gardés en tampon clavier et donnés à la prochaine lecture.



The image shows a C program in a text editor and its execution in a command prompt. The program, named `calculePerimetreRectangle.c`, uses `scanf` to read the length and width of a rectangle. The execution window shows the user entering '2' for the length, followed by three spaces, and then '0' for the width. The program outputs the perimeter as 4.00, demonstrating that `scanf` stopped at the first space and the remaining characters are still in the input buffer.

```
calculePerimetreRectangle.c X
#include <stdio.h>
int main ()
{
    float longueur, largeur, perimetre;
    printf ("Entrez la longueur du rectangle\n");
    scanf ("%f", &longueur);
    printf ("Entrez la largeur du rectangle\n");
    scanf ("%f", &largeur);
}
```


```
D:\mesCours\2013-2014\M1102\calculePerimetreRectangle.exe
Entrez la longueur du rectangle
2 000
Entrez la largeur du rectangle
Le rectangle de largeur 0.00 et de longueur 2.00 a un perimetre de 4.00_
```

# Limites du scanf

La lecture d'une donnée au format caractère « %c » doit être précédée d'un nettoyage du tampon clavier s'il a déjà servi ( pour ne pas récupérer la caractère de saut de ligne de fin de saisie précédente) .

```
#include <stdio.h>
int main ()
{
    int age ; char genre ; float taille ;
    printf ("Entrez votre age\n");
    scanf("%i", &age); // nettoyage à cause de cette saisie

    fflush(stdin);
    printf ("Tapez F pour Femme H pour Homme\n");
    scanf("%c", &genre);
```



# Afficher un texte : printf

A l'aide de la fonction printf .

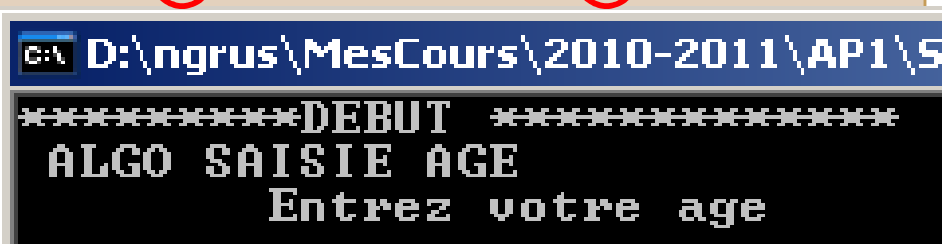
Un seul paramètre peut suffire : le texte à afficher !

```
printf ( " texte à afficher " );
```

Et des caractères spéciaux pour un peu de mise en forme.

Mise en forme	Caractères à insérer
Saut de ligne	\n
Tabulation	\t

```
printf("*****DEBUT *****\n ALGO SAISIE AGE\n");  
printf("\tEntrez votre age \n");
```



```
D:\ngrus\MesCours\2010-2011\AP1\5  
*****DEBUT *****  
ALGO SAISIE AGE  
Entrez votre age
```

# Afficher un texte : printf

Les caractères spéciaux doivent être précédés par \

Caractères spéciaux	Caractères à insérer
"	\"
\	\\

```
Le titre du film est " Bienvenue chez les roses"
```

```
printf("Le titre du film est \" Bienvenue chez les roses\" ");
```

```
Le chemin est c:\mesDocs\
```

```
printf("Le chemin est c:\\mesDocs\\");
```



# Afficher une variable: printf

A l'aide de la fonction **printf**.

**Prototype** : `int printf(const char * format, ...);`

Elle attend dans ce cas au moins 2 paramètres :

- Format pour afficher
- La variable à afficher

`printf( "%i" , largeur );`



Parametre 1 : format  
d'affichage de la  
valeur passée au sein  
du paramètre 2

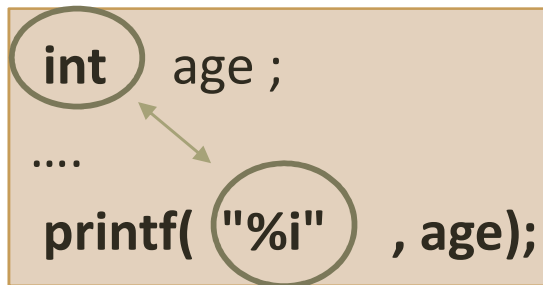


Parametre 2 : variable  
contenant la valeur à  
afficher

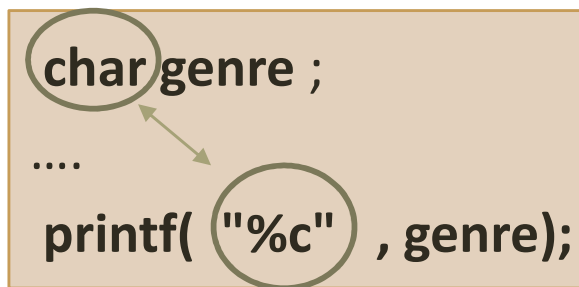
# Afficher une variable: printf

Le format doit être compatible avec le type déclaré !

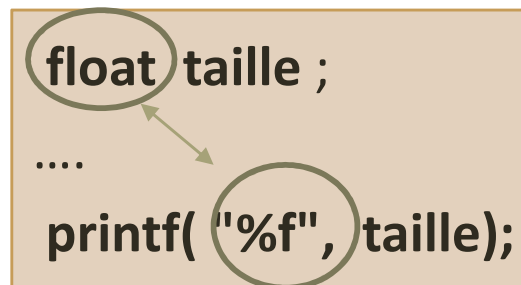
```
int age ;  
....  
printf( "%i" , age);
```



```
char genre ;  
....  
printf( "%c" , genre);
```



```
float taille ;  
....  
printf( "%f", taille);
```



# Formater l'affichage d'un réel

On peut facilement préciser le nombre de chiffre après la virgule.

```
#include <stdio.h>
```

```
int main ()
```

```
{
```

```
float nbf = 2.454 ;
```

```
printf ( " %f \n" , nbf );
```

```
2.454000
```

```
printf ( " %.2f \n " , nbf );
```

```
2.45
```

```
printf ( " %5.0f \n" , nbf );
```

```
2
```

# Afficher un texte avec une ou plusieurs variables

Introduire autant de formats que de variables à afficher.

```
printf("Vous avez %i ans et vous mesurez %f cm", age, taille );
```

2<sup>ème</sup> format => 2<sup>ème</sup> paramètre

1<sup>er</sup> format => 1<sup>er</sup> paramètre

# Faire des tests en C

# Structure alternative simple C

## Syntaxe:

```
if ( condition )  
{  
    instructions ;  
}  
else  
{  
    instructions;  
}
```

## Exemple :

```
if (moyenne < 10 )  
{  
    printf (" Vous n'avez pas eu la moyenne.");  
}  
else  
{  
    printf(" Vous avez au moins la moyenne.");  
}
```

*Remarque : S'il n'y a qu'une seule instruction, les accolades ne sont pas nécessaires.*

# Structure alternative simple C

Le bloc else n'est pas obligatoire.

Syntaxe:

```
if( condition )  
{  
    instructions;  
}
```

Exemple :

```
if (note < 10 )  
{  
    printf (" Vous irez au soutien.");  
}
```

Exemple :

```
if (note < 10 )  
    printf (" Vous irez au soutien.");
```

# Conditions multiples C

- Et : &&
- Ou : ||

Exemples :

```
if(heure >= 12 && heure <14 )
```

```
    printf ( " pause repas ");
```

```
if(heure >= 22 || heure <6 )
```

```
    printf ( " pause sommeil ");
```



# Autre structure : Switch

## Syntaxe:

```
switch ( var)
{
    case valeur1 : instructions;
                    break;
    case valeur2 : instructions ;
                    break;
    ...
    default : instructions;
}
```

## Exemple:

```
char note;
printf("Entrez A,B,C,D\n");
scanf("%c",&note);
switch ( note)
{
    case 'A' : printf("très bien") ;
                break;
    case 'B' : printf("bien") ;
                break;
    case 'C' : printf("médiocre") ;
                break;
    case 'D' : printf("insuffisant") ;
                break;
    default : printf("erreur !") ;
}
```

# Imbriquer

Deux syntaxes sont parfois possibles :

Exemple:

```
if (chiffre >0)
    { printf( " chiffre positif " );}
else
{
    if (chiffre <0)
        {printf( " chiffre négatif " );}
    else
        { printf( " chiffre nul " );}
}
```

Exemple:

```
if (chiffre >0)
    { printf( " chiffre positif " );}
else if (chiffre <0)
    {printf( " chiffre négatif " );}
else
    { printf( " chiffre nul " );}
```

# Notion de portée de variable

Portée d'une variable : une variable n'existe que dans le bloc (et blocs imbriqués) où elle est a été déclarée.

Exemple contenant un problème lié à la portée des variables :

```
int main()
{
    float moyenne;
    ....
    if (moyenne < 10)
    {
        printf(" \n Voulez vous valider le semestre ? ");
        fflush(stdin);
        char validation;
        scanf ( "%c", &validation);
    }
    if ( validation == 'O' || moyenne >= 10)
    { printf ( "Semestre valide" ); }
}
```

Pb ! Instruction appartenant au bloc main, validation n'a pas été définie au sein du main

# Répéter des actions en C

# Structure DO...WHILE

## Syntaxe:

```
do
{
    instructions ;
} while (condition);
```

Attention : ne pas oublier ;

## Exemple :

```
char recommence ;
do
{
    // instructions du programme .....
    printf(" Recommencez (O/N) ");
    fflush(stdin);
    scanf (" %c" , &recommence) ;
} while ( recommence == 'O');
```

Quand il n'y a qu'une seule instruction les {} ne sont pas obligatoires

# Structure DO...WHILE

Faire une boucle volontairement infinie peut être utile pour un jeu !

Exemple :

```
do  
{  
    // instructions du jeu.....  
} while (1);
```

# Structure WHILE

## Syntaxe:

```
while (condition)
{
    instructions ;
}
```

Attention : ne pas mettre;

## Exemple :

```
char type ;
printf( " votre type (F/H) ");
fflush(stdin);
scanf ( " %c" , &type) ;
while( ( type != 'H') && ( type != 'F') )
{
    printf( " Erreur de saisie . Vous devez saisir F ou H : ");
    fflush(stdin);
    scanf ( " %c" , &type) ;
}
```

# ! Le non logique

## Exemple :

```
char type ;  
printf( " votre type (F/H) ");  
fflush(stdin);  
scanf ( " %c" , &type) ;  
while( ! ( type == 'H' || type == 'F') )  
{  
    printf( " Erreur de saisie . Vous devez saisir F ou H : ");  
    fflush(stdin);  
    scanf ( " %c" , &type) ;  
}
```



# Structure FOR

## Syntaxe:

```
for ( initialisation ; condition de continuité ; pas )  
{  
    instructions;  
}
```

## Exemple :

```
int i ;  
for ( i=1 ; i <= 10 ; i = i + 1 )  
{ printf ("%i",i); }
```

```
int i ;  
for ( i=1 ; i <= 10 ; i++ )  
    { printf ("%i",i); }
```

# Structure FOR

- Les variables de boucle peuvent être multiples.
- La condition de continuité peut être multiple assemblée par des opérateurs logiques.

Exemple : on donne 3 chances à un enfant pour donner un chiffre pair

```
int i,pair;
```

```
for( i=1,pair=0 ; i<=3 && pair == 0 ; i++ )  
{  
    int val ;  
    printf ("entre une valeur paire ");  
    scanf("%i",&val);  
    if(val%2==0)  
    { printf ("Bien");  
      pair = 1;  
    }  
    else  
        printf ("Non, %i n'est pas une valeur paire ." , val);  
}
```

# Un tableau

Un tableau sert à stocker un ensemble de valeurs de même nature. Elles sont stockées de manière contiguë au sein de la mémoire.

Représentation conceptuelle

0	1	2	3	4	5	...	29
10	15	20	12	14			

Représentation « physique »

10
15
20
12
14

# Déclarer un tableau

## Syntaxe :

type nomVariable [ taille ];

## Exemple :

int notes [30] ;

## Exemple :

float notes [30] ;

indices

0	1	2	3	4	5	...	29

1 élément du tableau

# Affecter une case

Syntaxe :

**nomVariable [ indice ] = valeur;**

Exemple :

notes [0] =20 ;

notes [1] =15 ;

0	1	2	3	4	5	...	29
20	15						

# Initialiser un tableau

Exemple d'une initialisation par défaut :

```
int notes [30] ;  
int i;  
for (i = 0 ; i <=29 ;i ++)  
    notes[i] = 0 ;
```

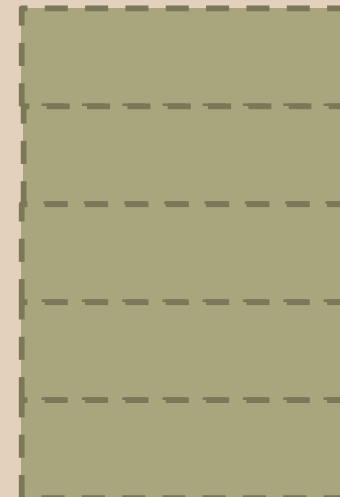
Exemple avec des saisies utilisateur :

```
int notes [30] ;  
int i;  
for (i = 0 ; i <=29 ;i ++)  
    { printf ( " Entrez la note %i \n ", (i+1));  
      scanf ( "%i", &notes[i] );  
    }
```

&notes[0]

&notes[1]

&notes[2]



# Afficher un tableau

Exemple :

```
int notes [30] ;
```

```
int i;
```

```
for (i = 0 ; i <=29 ;i ++)
```

```
    printf ( "%i \n", notes[i ] );
```

# Utiliser des Fonctions/Procédures en C



# Notion de librairie

- Le langage C comprend un certain nombre de fonctions déjà définies, prêtes à l'emploi. Elles sont regroupées dans la librairie standard.
- Il existe des fichiers d'entêtes ( de signatures ) de toutes ces fonctions . *Ex : math, ctype, stdio, stdlib, string, time ...*

# Notion de librairie

- Extrait du site : [www.cplusplus.com/reference](http://www.cplusplus.com/reference)

Reference

C library:

<cassert> (assert.h)

<cctype> (ctype.h)

<cerrno> (errno.h)

<cfenv> (fenv.h)

<cmath> (math.h)

<cinttypes> (inttypes.h)

<ciso646> (iso646.h)

<climits> (limits.h)

<locale> (locale.h)

<cmath> (math.h)

<csetjmp> (setjmp.h)

<csignal> (signal.h)

<cstdarg> (stdarg.h)

<cstdbool> (stdbool.h)

<cstddef> (stddef.h)

<cstdint> (stdint.h)

<stdio> (stdio.h)

<stdlib> (stdlib.h)

<string> (string.h)

<tgmath> (tgmath.h)

<ctime> (time.h)

<cuchar> (uchar.h)

<wchar> (wchar.h)

header

## <cmath> (math.h)

---

### C numerics library

Header <cmath> declares a set of functions to compute common mathematical operat

---

### Functions

---

#### Trigonometric functions

cos	Compute cosine (function )
sin	Compute sine (function )
tan	Compute tangent (function )
acos	Compute arc cosine (function )
asin	Compute arc sine (function )
atan	Compute arc tangent (function )
atan2	Compute arc tangent with two parameters (function )

---

#### Hyperbolic functions

cosh	Compute hyperbolic cosine (function )
sinh	Compute hvnerbolic sine (function )

# Signature d'une fonction

typeRetour **nomFonction** (type param1, type param2)

**double pow (double base, double exponent);**

Raise to power

Returns *base* raised to the power *exponent*:  $\text{base}^{\text{exponent}}$

**void nomProcedure** (type param1, type param2)

**void rectangle (int left, int top, int right, int bottom);**

rectangle draws a rectangle in the current line style, thickness, and drawing color.

# Utiliser une fonction

Pour utiliser une fonction, il faut :

- Respecter sa signature
- Mentionner le lien vers son fichier d'entête

```
double pow (double base, double exponent);
```

```
#include <math.h>
```

```
int main( )
```

```
{
```

```
    double res = pow(2,3);
```

```
    printf("%.0f\n",res);    //8
```

```
}
```

Le type double étant le plus grand des numériques, il accepte tous les autres numériques (float, int ...)

# Particularité

Certaines fonctions en C ont un nombre de paramètre(s) variable(s).

```
int printf ( const char * format, ... );
```

Print formatted data to stdout

Writes the C string pointed by *format* to the standard output (stdout). If *format* includes *format specifiers* (subsequences beginning with %), the additional arguments following *format* are formatted and inserted in the resulting string replacing their respective specifiers.

```
#include <stdio.h>
```

```
int main( )
```

```
{ int a = 3, b= 4;
```

```
printf("test");
```

```
printf("valeur de a : %i\n", a);
```

```
printf(" %i + %i\n", a, b);
```

2 paramètres

1 paramètre

3 paramètres

# Particularité

On doit souvent exploiter le résultat d'une fonction,  
ce n'est pas pour autant obligatoire ...

**int printf ( const char \* format, ... );**

**Return Value :** On success, the total number of characters written is returned. If a writing error occurs, the *error indicator* (error) is set and a negative number is returned.

```
#include <stdio.h>
int main( )
{
    int r =printf("test");
    printf("\nresultat de printf : %i\n",r);
}
```

```
test
resultat de printf : 4
```

# Retour du scanf...

```
int scanf ( const char * format, ... );
```

## Return Value

On success, the function returns the number of items of the argument list successfully filled. This count can match the expected number of items or be less (even zero) due to a matching failure, a reading error.

```
printf("Entrez un entier");
res = scanf("%i",&chiffre);
while( res!=1)
{
    printf("Erreur. Entrez un entier :\n");
    fflush(stdin);
    res = scanf("%i",&chiffre);
}
printf("entier : %i \n",chiffre);
```

```
printf("caractère compris entre a et z") ;
res=scanf("%1[a-z]c", &rep);
while (res !=1)
{
    printf("caractère compris entre a et z") ;
    fflush(stdin);
    res=scanf("%1[a-z]c", &rep);
}
```

# Pourquoi ?

- ⇒ Pour éviter de dupliquer du code dans un programme
- ⇒ Pour limiter le nombre de lignes de code dans l'algorithme principal et améliorer sa lisibilité
- ⇒ Pour créer ses propres fonctions « maison » et les réutiliser à volonté
- ⇒ Pour découper un algorithme et travailler ainsi à plusieurs



# Définir des Fonctions/Procédures en C

# Procédure sans paramètre

```
void nomProcedure ( )  
{  
    instruction ;  
    ....  
}
```

```
void menu( )  
{    printf (« + pour additionner ») ;  
    ...  
}
```

```
int main ( )  
{menu() ;  
... }
```

# Procédure avec paramètre(s)

```
void nomProcedure ( type param1, type param2,...)
{
    instruction ;
    ....
}
```

```
void afficheAddition( int a , int b )
{   printf (« %i + %i = %i » , a, b, (a+b)) ; }
```

```
int main ()
{afficheAddition( 2,4) ;
... }
```

# Fonction sans paramètre

```
typeRetour nomFonction ( )  
{  
    instruction ;  
    ....  
    return ..... ;  
}
```

```
char menu( )  
{ char ope ;  
    printf (« + pour additionner ») ; ...  
    return ope;  
}
```

```
int main ()  
{ char operateur;  
    operateur= menu( ) ;  
    ... }
```

# Fonction avec paramètre(s)

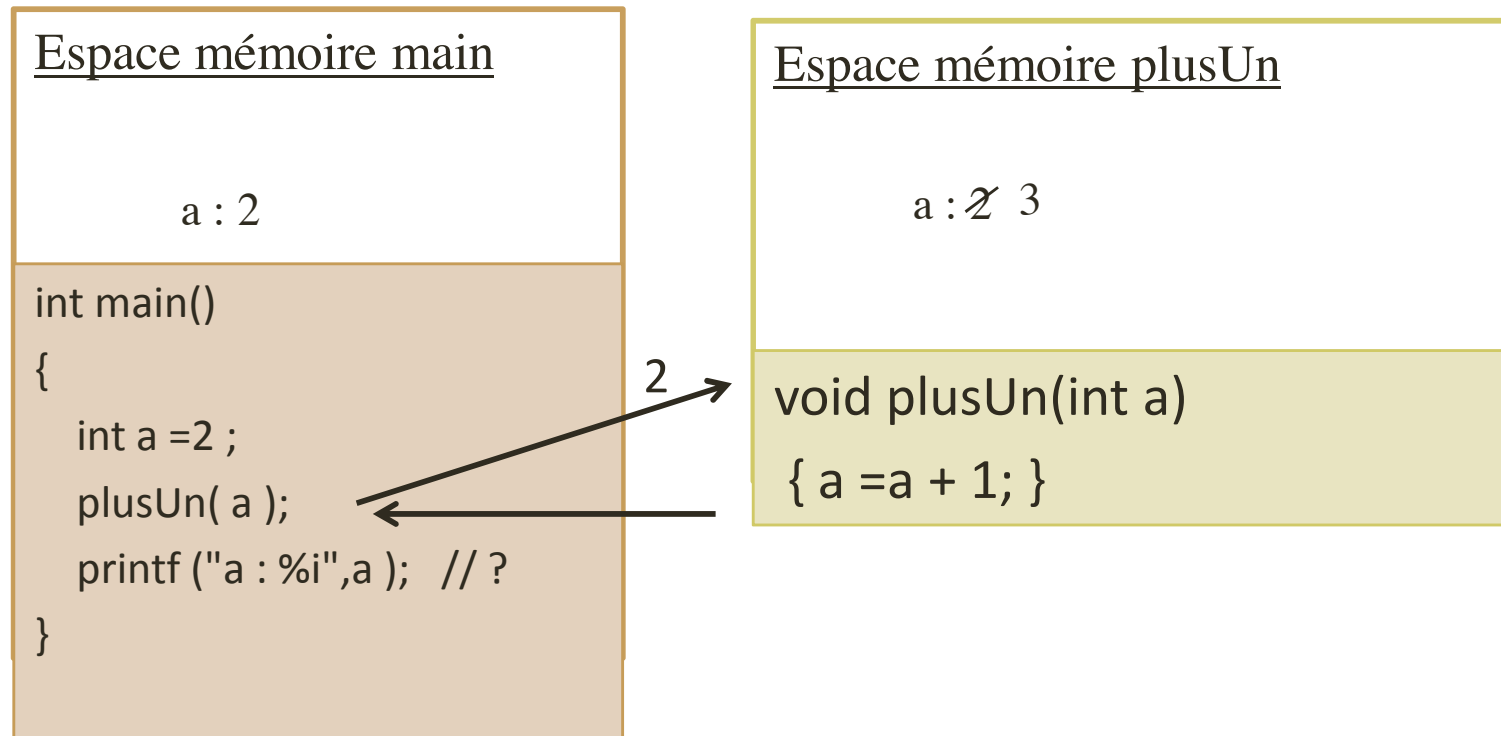
```
typeRetour nomFonction ( type param1, type param2,... )  
{  
    instruction ;  
    ....  
    return ..... ;  
}
```

```
int addition( int a , int b)  
{ int res = a + b ;  
    return res;  
}
```

```
int main ()  
{ int res ;  
    res = addition ( 2,4) ;  
    ... }
```

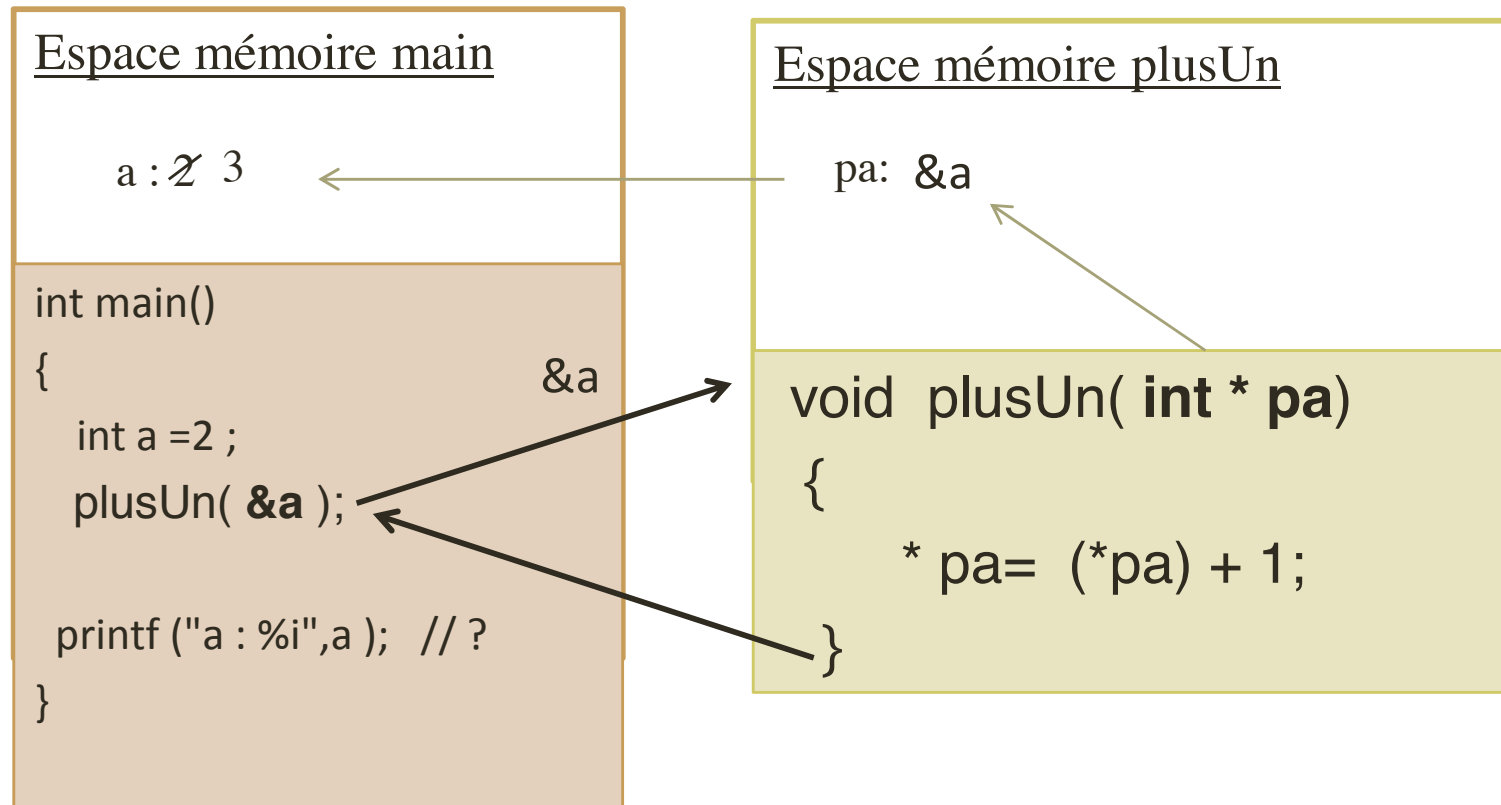
# Passage par valeur

Par défaut tous les paramètres, à l'exception des tableaux sont passés par valeur. Dans un sous programme, on travaille avec une copie de la valeur passée.



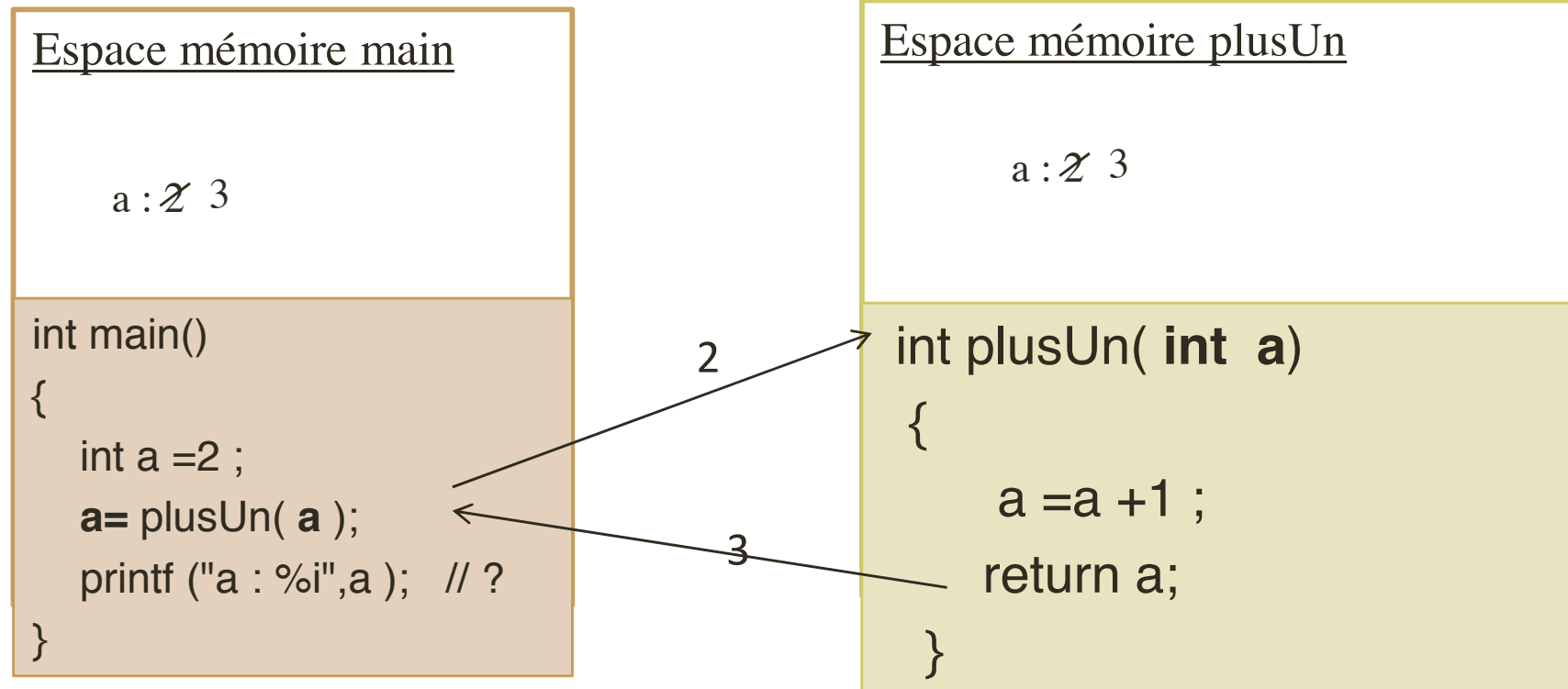
# Passage par adresse

On peut spécifier un passage par adresse.



# Passage par adresse

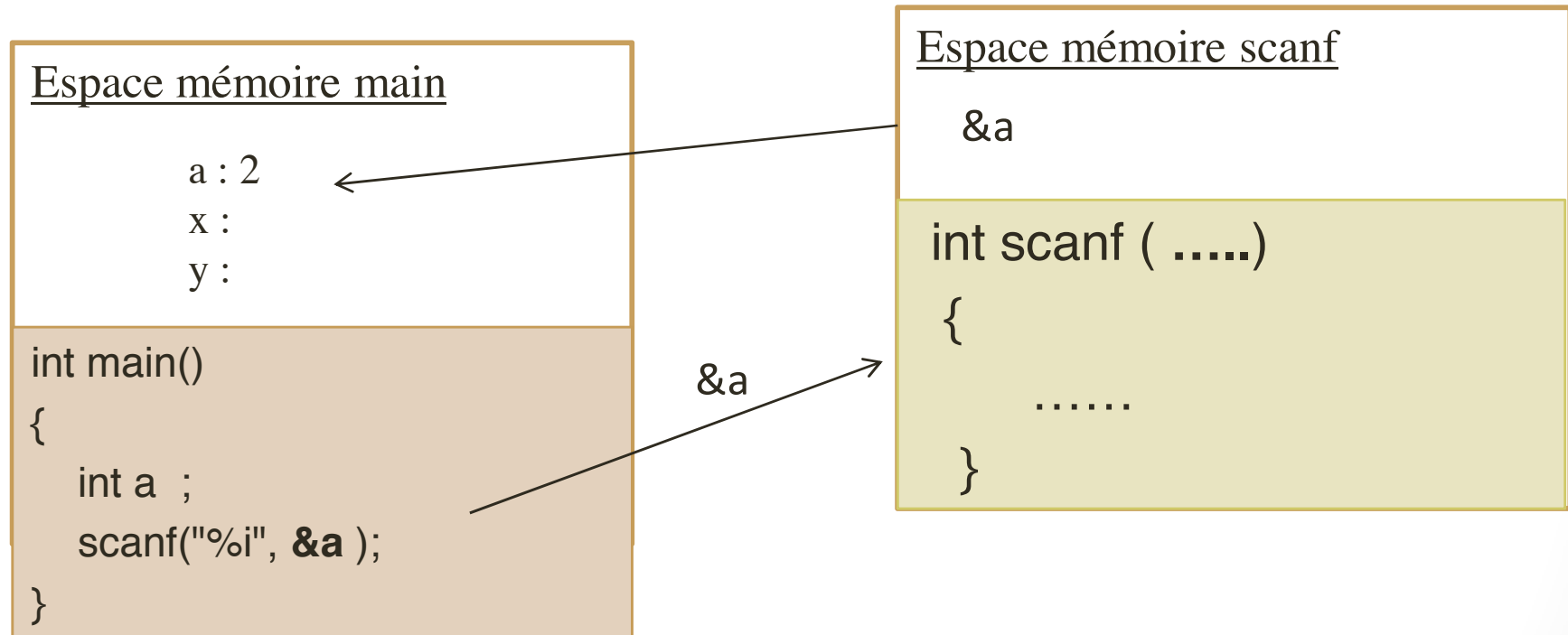
On peut souvent s'en passer .....





# Passage par adresse

Un exemple bien connu ...



# Passage par adresse

Parfois, utiliser pour « retourner » plusieurs valeurs .... Comme c'est impossible, on écrit dans la mémoire du programme appelant les différents résultats.

Pas très beau tout ça !

## Espace mémoire main

a : 2

x :

y :

```
int main()
{
    initWindow(200,200);
    int x,y ;
    getmouseclick ( &x,&y);
}
```

## Espace mémoire getmouseclick

&x &y

```
void getmouseclick (int kind,
                    int * x, int * y)
```

```
{
```

.....

```
}
```

&x, &y

# Passage d'un tableau en paramètre

Le tableau est par défaut passé par adresse.

Espace mémoire main

0	1		19

```
int main()
{
    int const TAILLE_MAX = 20;
    int notes [TAILLE_MAX] ;
    initTableau ( notes , TAILLE_MAX ) ;
}
```

```
void initTableau(int notes [ ], int taille)
{
    int i ;
    for ( i = 0 ; i <= taille; i++ )
        notes[ i ] = 0;
}
```

# Où définir les sous-programmes

Un sous-programme doit être déclaré avant d'être utilisé.

```
int addition( int a , int b)
{  int res = a + b ;
   return res;
}
```

```
int main( )
{
  int nb1=12,nb2=23;
  int r = addition(nb1,nb2);
  printf("%i\n",r);
  system("pause");
  return 0;
}
```

# Où définir les sous-programmes

On préférera ne mettre que les entêtes (signatures) des fonctions au début pour atteindre le cœur du programme plus vite

```
int addition( int a , int b ) ;  
int main( )  
{  
    int nb1=12,nb2=23;  
    int r = addition(nb1,nb2);  
    ...  
}  
int addition( int a , int b)  
{ int res = a + b ;  
  return res;  
}
```

# Où définir les sous-programmes

**Remarque : il est possible de découper notre code physiquement**

calcul.h

```
int addition( int a , int b ) ;
```

Calcul.c

```
int addition( int a , int b )  
{ int res = a + b ;  
  return res;  
}
```

main.c

```
#include "calcul.h"  
int main(int argc, char *argv[])  
{  
    int nb1=12,nb2=23;  
    int r = addition(nb1,nb2);  
    printf("%i\n",r);  
    system("pause");  
  
    return 0;  
}
```



# Quelques règles ...

Les règles d'écriture de fonctions ou procédures en C sont les mêmes que celles données en algorithmie :

- Pour communiquer des valeurs contenues dans le programme principal aux fonctions ou procédures, il faut définir des paramètres
- Pour communiquer la valeur « résultat » contenue dans une fonction au programme principal , il faut utiliser l'instruction de retour
- Les variables sont locales aux sous-programmes.
- Il faut éviter les imbrications d'appel.