Fonctions en C

Encapsule un traitement particulier formant un tout

- Peut implémenter la notion de module en logique
- Notion de librairie
- Augmente la lisibilité d'un programme
- Réalise un objectif précis
- Améliore le débogage et la maintenance d'un programme

Son utilisation se décompose en trois phases :

- Définition de la fonction
- Déclaration de la fonction
- Appel de la fonction

Déclarer une fonction

Appeler une fonction

Règles de visibilité des variables

Passage des paramètres par valeur

Fonction renvoyant une valeur au programme

Passage des paramètres par valeur et par adresse

Passage des tableaux aux fonctions

Fonctions en C

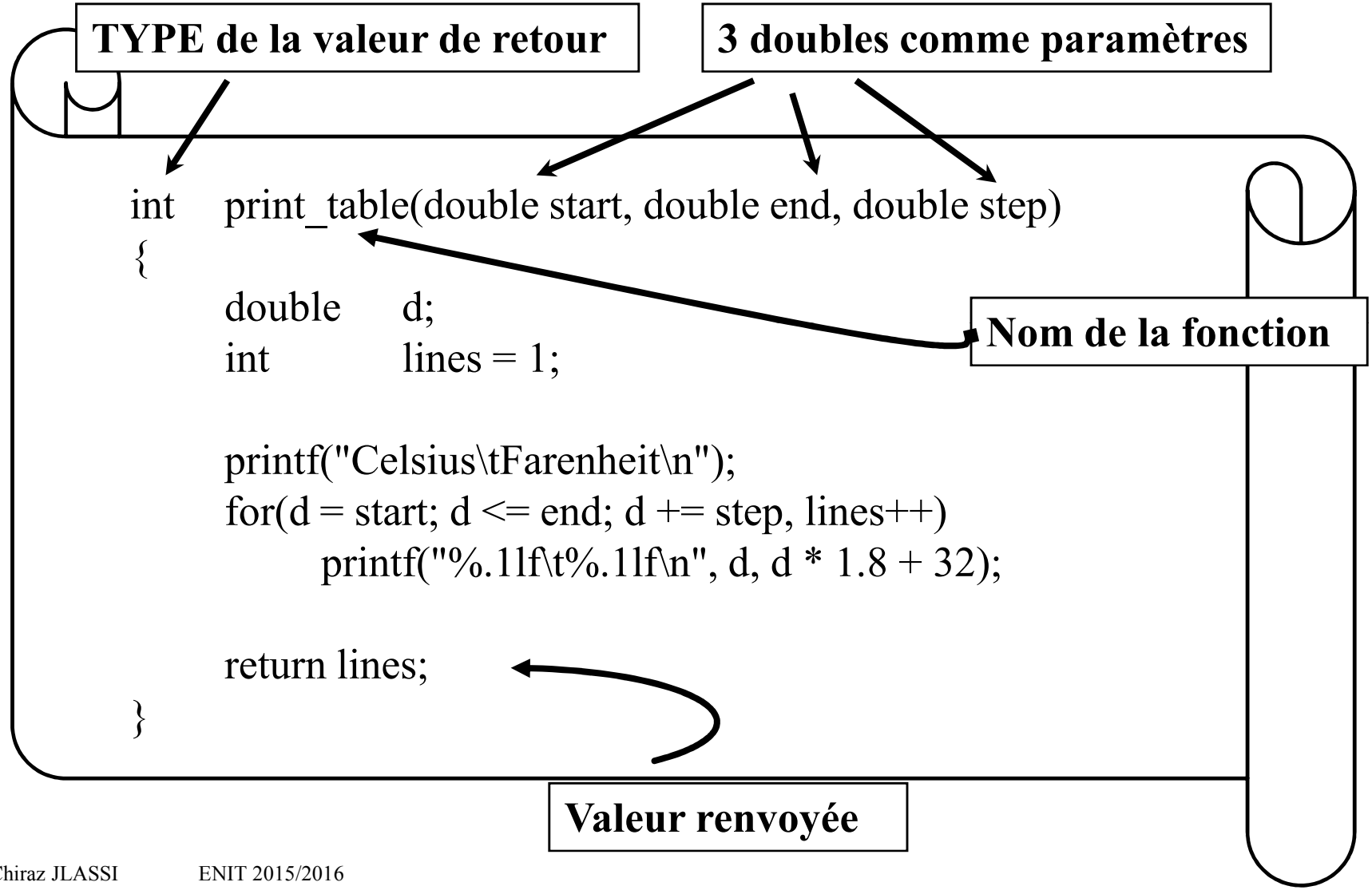
Les principes

- En C, tout est fonction (Définition, Déclaration et Appel).
- Une fonction peut avoir autant de paramètres que l'on veut, même aucun

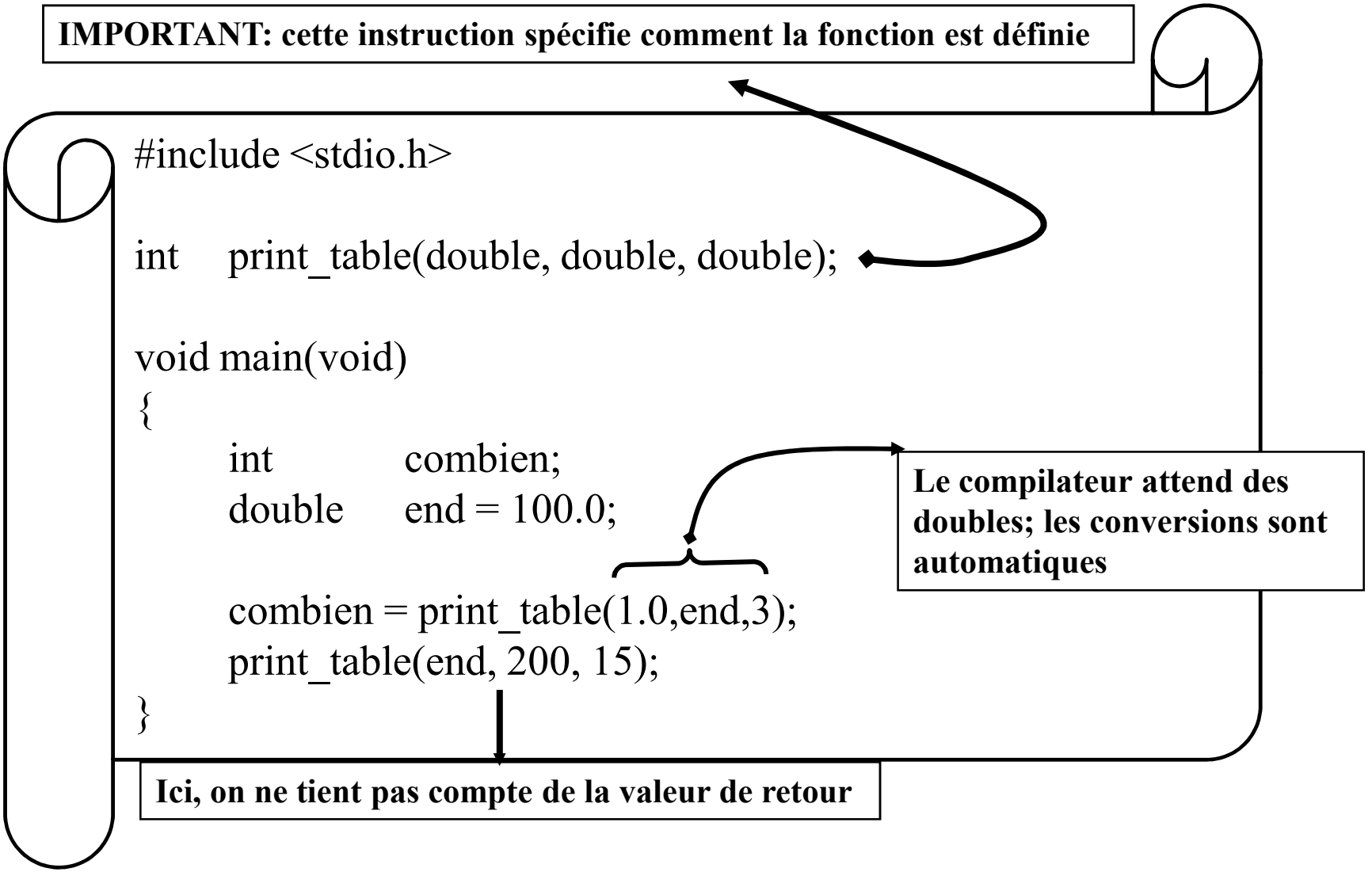
  (comme void main(void))
- Une fonction renvoie une valeur ou aucune.
- Les variables déclarées dans une fonction sont locales à cette fonction
- **Une fonction possède un et un seul point d'entrée, mais éventuellement plusieurs points de sortie (à l'aide du mot return).**
- **L'imbrication de fonctions n'est pas autorisée:**

  une fonction ne peut pas être déclarée à l'intérieur d'une autre fonction. Par contre, une fonction peut appeler une autre fonction. Cette dernière doit être déclarée avant celle qui l'appelle.

Déclarer une fonction



Appeler une fonction



```
   float calcule(float, float);
   int addition();
```
Déclaration

```
void main(void)
{
 int Val ;

 Val = addition();
  printf("val = %d", Val);
}
```
Appel

```
int addition()
{
 float tmp;
   tmp = calcule(2.5,3) + calcule(5,7.2);
  return (int)tmp;
}

float calcule(float float1, float float2)
{
 return (  (float1 + float2 ) / 2) ;
}
```
Définition

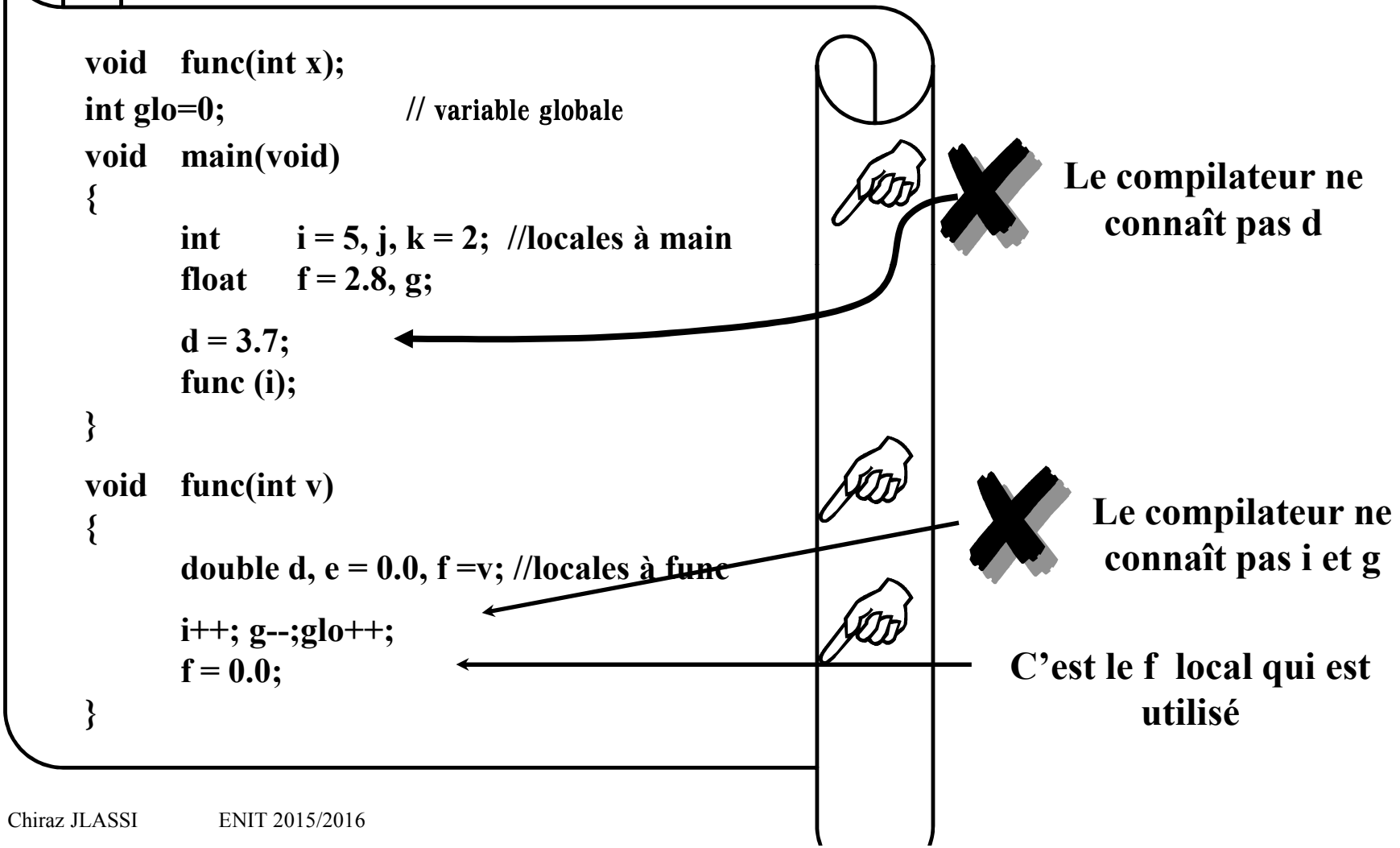
Règles de visibilité des variables

Le C est un language structuré en blocs { } , les variables ne peuvent être utilisées que là où elles sont déclarées

```
void   func(int x);
int glo=0;                  // variable globale
void   main(void)
{
       int      i = 5, j, k = 2;  //locales à main
       float    f = 2.8, g;

       d = 3.7;
       func (i);
}
void   func(int v)
{
       double d, e = 0.0, f =v; //locales à func

       i++; g--;glo++;
       f = 0.0;
}
```



Passage des paramètres par valeur

* Quand une fonction est appelée, ses paramètres sont copiés (passage par valeurs)

* La fonction travaille donc sur des copies des paramètres et ne peut donc les modifier

* En C, le passage des paramètres par référence se fait en utilisant des pointeurs (voir plus loin)

* scanf() travaille avec pointeur. C'est le pourquoi du &

Passage des paramètres par valeur

```
#include <stdio.h>

void   change(int v);

void   main(void)
{
        int var = 5;

        change(var);

        printf("main: var = %d\n", var);
}

void   change(int v)
{
        v *= 100;
        printf("change: v = %d\n", v);
}
```

```
change: v = 500
main: var = 5
```

FONCTION RENVOYANT UNE VALEUR AU PROGRAMME

```
#include <stdio.h>

int     change(int v);

void    main(void){
        int var = 5;
        int valeur;
        valeur = change(var);

        printf("main: var = %d\n", var);
        printf("main: valeur = %d\n", valeur);
}

int change(int v)
{
        v *= 100;
        printf("change: v = %d\n", v);
        return (v+1);
}
```

```
change: v = 500
main: var = 5
main: valeur = 501
```

Une fonction se termine et « rend la main » au code appelant lorsque son exécution rencontre l'instruction : return expression; ou return;

```
#include <stdio.h>

int return_Val(int v);

void   main(void){
        int var = 5;
        int valeur;
        valeur = return_Val(var);

        printf("main: var = %d\n", var);
        printf("main: valeur = %d\n", valeur);
}

int return_Val(int v)
{
        if (v == 10)        return (2*v);
        else                return (3*v);
}
```

```
main: var = 5
main: valeur = 15
```

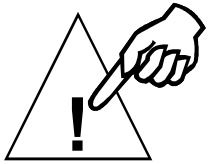
```
int max (int a, int b) {
        if (a > b)        return a ;
        else return b ;
}
int min (int a, int b) {
        if (a < b)       return a ;
        else return b ;
}
int difference (int a, int b, int c) {
     return (max (a, max(b,c)) - min (a, min (b, c))) ;
}
void main ()
{
   printf ("%d", difference (10, 4, 7)) ;
}
```

La Récursivité

```
int fact( int n) {
        if (n < 1) return 1;
        else return ( n * fact(n-1) );
}
```

```
void hanoi(int n, int a, int
b){
int c ;
…
    if (n == 1) move(a,b);
    else {
        hanoi(n-1, a, c);
        move (a, b);
        hanoi(n-1, c, b);
    }
}
```

Passer des tableaux aux fonctions



- Les tableaux peuvent être passés comme paramètres d'une fonction.
- Ils ne peuvent pas être retournés comme résultat d'une fonction.
- La longueur du tableau ne doit pas être définie à la déclaration de la fonction.
- Un tableau peut être modifié dans une fonction. Il est passé par référence (adresse) et non par valeur.

```
#include <stdio.h>
void    somme(int x[], int n);
void    main(void){
        int i;
        int p[6] = { 1, 2,3, 5, 7, 11 };
        somme(p, 6);
        for (i=0;i<6;i++)
                printf("%d ", p[i]);
}

void    somme(int a[], int n){
        int     i;
        for(i = n-1; i >0 ; i--)
                a[i] += a[i-1];
}
```

p avant somme

1
2
3
5
7
11

p après somme

1
3
5
8
12
18

Exemple: transposition d'une matrice

```
void transpose ( float a[][], int m, int n);

void main(void)
{float matrice [100][20];   /* matrice (100,20) */
 /*…*/
 transpose (matrice, 3, 2);
}

void transpose ( float a[][], int m, int n)
{ int i,j;
  float tmp;
    for (i=0;i<m;i++)
      for(j=0;j<i;j++) {
            tmp = a[i][j];
            a[i][j]=a[j][i];
            a[j][i]=tmp;
      }
  }
}
```

LES ALGORITHMES DE TRI ET DE RECHERCHE