

# Les fonctions le retour

30



# La semaine dernière

31

- Les fonctions
- Les appels de fonctions
- Les passages de paramètres
- Les valeurs de retour

# Sommaire

32

- Le polymorphisme paramétrique
- Les valeurs par défaut
- La portée des variables
- Le passage par référence
  - tableau
  - enregistrement
- Architecture d'un projet multi-fichiers

# Le polymorphisme paramétrique

33

```
// calcul le max de 2 entiers
// una : un entier
// unB : un autre entier
// retourne le max de unA, unB
int max (int unA, int unB)
{
    return unA>unB ? unA:unB ;
}
```

```
int main()
{
    std::cout << max(1,3) << std::endl;
    std::cout << max(3,2) << std::endl;
    std::cout << max(1,2,3) << std::endl;
    std::cout << max(2,3,1) << std::endl;
    std::cout << max(3,2,1) << std::endl;
}
```

R1.01 DEV1 - Initiation au développement

```
// calcul le max de 3 entiers
// una : un entier
// unB : un autre entier
// unC : encore un autre entier
// retourne le max de unA, unB et unC
int max (int unA, int unB, int unC)
{
    return max(max(unA, unB), unC) ;
}
```

La différence se fait sur la signature  
de la fonction : les types passés en  
paramètre.

Elle ne peut se faire seulement sur la  
valeur de retour.

ECa - JMBo - MaHa - RCh

# Les valeurs par défaut

34

```
1  #include <iostream>
2  #include <climits>
3
4  // calcul le max de 2 ou 3 entiers
5  // una : un entier
6  // unB : un autre entier
7  // unC : encore un autre entier facultatif
8  // retourne le max de unA, unB et unC
9  int max (int unA, int unB, int unC=INT_MIN)
10 {
11     int unX = unA>unB ? unA:unB ;
12     return unC>unX ? unC:unX ;
13 }
14
15 int main()
16 {
17     std::cout << INT_MIN << std::endl;
18     std::cout << max(1,3) << std::endl;
19     std::cout << max(3,2) << std::endl;
20     std::cout << max(1,2,3) << std::endl;
21     std::cout << max(2,3,1) << std::endl;
22     std::cout << max(3,2,1) << std::endl;
23 }
```

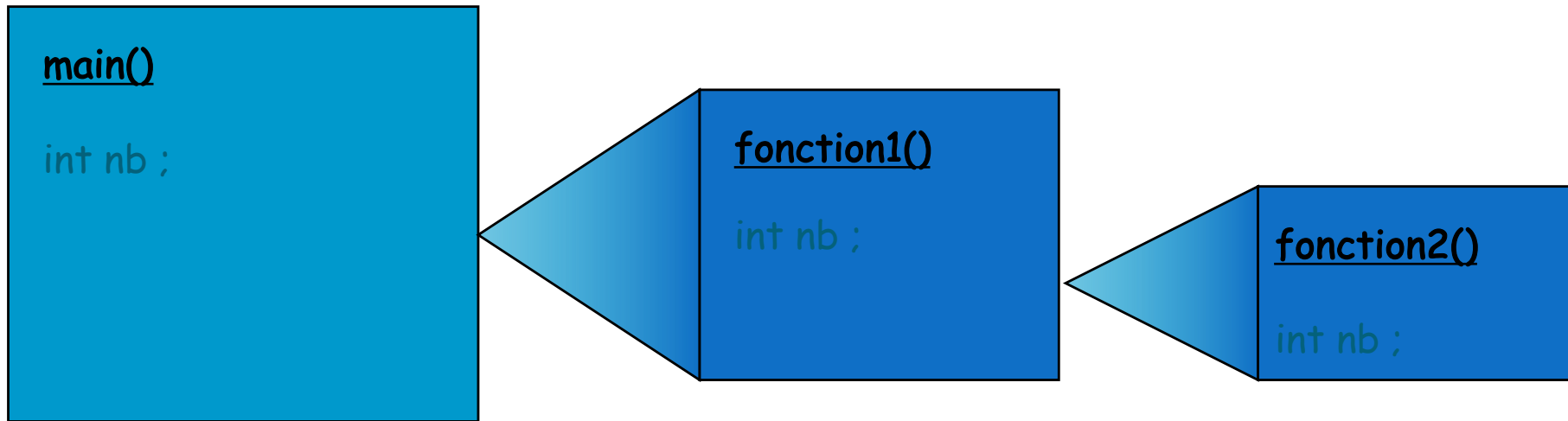
On peut définir un ou plusieurs  
paramètres par défaut en  
commençant toujours par les  
derniers.

-2147483648

3  
3  
3  
3  
3

# La portée des variables

35



- La visibilité de chaque variable `nb` est limitée à chacune des fonction dans laquelle elle est définie.
- La durée de vie de chacune des variables `nb` est égale à la durée de vie de l'exécution de la fonction dans laquelle elle est définie.

# La portée des variables

36

```
3 void fonction()  
4 {  
5     int leN = 0 ;  
6  
7     for(int leI=0 ; leI<10 ; leI++)  
8     {  
9         int leN = leI ;  
10        int leP = 5 ;  
11        std::cout << leN << leP << leI << std::endl ;  
12    }  
13    std::cout << leN << leP << leI << std::endl ;  
14  
15    {  
16        int leN = 100 ;  
17        int leP = 0 ;  
18        std::cout << leN << leP << leI << std::endl ;  
19    }  
20    std::cout << leN << leP << leI << std::endl ;  
21 }
```

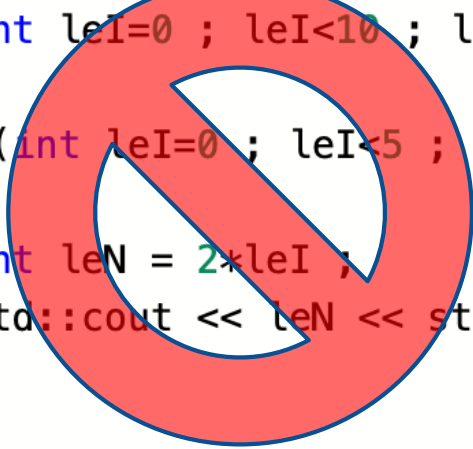
La portée ou visibilité des variables est limitée au bloc d'instructions dans lequel elle est déclarée.

NB : leI appartient au for.

On peut déclarer une variable locale du même nom qu'une variable locale d'un bloc englobant mais cela peut porter à confusion.

# La portée des variables

37



```
for(int leI=0 ; leI<10 ; leI++)  
{  
    for(int leI=0 ; leI<5 ; leI++)  
    {  
        int leN = 2*leI ;  
        std::cout << leN << std::endl ;  
    }  
}
```

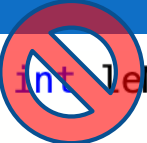
```
for(int leI=0 ; leI<10 ; leI++)  
{  
    for(int leJ=0 ; leJ<5 ; leJ++)  
    {  
        int leN = 2*leI ;  
        std::cout << leN << std::endl ;  
    }  
}
```

**C'est possible en C++ !!**



# Variables locales VS globales

38

```
3   int leN = 666 ;    variable globale hors de toute fonction
4
5  // WTF
6  void fonction()
7  {
8      int leN = 0 ;    variable locale à la fonction fonction()
9      std::cout << leN << std::endl ;
10     std::cout << ::leN << std::endl ;
11     ::leN = 111 ;
12 }
13
14 int main()
15 {
16     int leN = 100 ;    variable locale à la fonction main()
17     std::cout << leN << std::endl ;
18     std::cout << ::leN << std::endl ;
19     fonction() ;
20     std::cout << leN << std::endl ;
21     std::cout << ::leN << std::endl ;
22     return 0 ;
23 }
```

L'opérateur de résolution de portée

:: permet d'accéder à une variable définie dans un espace donné :  
ios::app, std::cout ou à faire référence à la variable globale. Son usage est à éviter sauf pour les définitions de constantes.

# Passage par valeur VS référence

39

```
// permute 2 nombres
// nb1 : le premier nombre à permuter
// nb2 : l'autre nombre à permuter
void swap(int nb1, int nb2)
{
    int temp = nb1 ;
    nb1 = nb2 ;
    nb2 = temp ;
}

int main()
{
    int nb1 = 10 ; int nb2 = 20 ;
    cout << "nb1 = " << nb1
         << ", nb2 = " << nb2 << endl ;
    swap(nb1, nb2) ;
    cout << "nb1 = " << nb1
         << ", nb2 = " << nb2 << endl ;
}
```

nb1 = 10, nb2 = 20  
nb1 = 10, nb2 = 20

```
// permute 2 nombres
// nb1 : le premier nombre à permuter
// nb2 : l'autre nombre à permuter
void swap(int& nb1, int& nb2)
{
    int temp = nb1 ;
    nb1 = nb2 ;
    nb2 = temp ;
}

int main()
{
    int nb1 = 10 ; int nb2 = 20 ;
    cout << "nb1 = " << nb1
         << ", nb2 = " << nb2 << endl ;
    swap(nb1, nb2) ;
    cout << "nb1 = " << nb1
         << ", nb2 = " << nb2 << endl ;
}
```

nb1 = 10, nb2 = 20  
nb1 = 20, nb2 = 10

# Les tableaux en paramètre

41

```
6 // display each element of an array with a space between and a carriage return at the end
7 // array : the array to display
8 // size : array size
9 void displayArray(int array[], int size)
10 {
11     for(int index=0 ; index<size ; index++)
12     {
13         cout << array[index] << " ";
14     }
15     cout << endl;
16 }
```

passage en paramètre  
du tableau et sa taille

```
18 // fill each element of an array with the index value
19 // array : the array to display
20 // size : array size
21 void fillIndexArray(int array[], int size)
22 {
23     for(int index=0 ; index<size ; index++)
24     {
25         array[index] = index;
26     }
27 }
```

Un tableau est passé par référence  
implicite → on travaille directement  
sur les éléments du tableau.

# Les tableaux en paramètre

42

```
6 // display each element of an array with a space between and a carriage return at the end
7 // array : the array to display
8 // size : array size
9 void displayArray(int array[], int size)
10 {
11     for(int index=0 ; index<size ; index++)
12     {
13         cout << array[index] << " ";
14     }
15     cout << endl;
16 }
17
18 // fill each element of an array with the index value
19 // array : the array to display
20 // size : array size
21 void fillIndexArray(int array[], int size)
22 {
23     for(int index=0 ; index<size ; index++)
24     {
25         array[index] = index;
26     }
27 }
```

```
72 int array[10];
73
74 displayArray(array, 10);
75 fillIndexArray(array, 10);
76 displayArray(array, 10);
```

```
43 1 256 0 0 0 20 -121 4 76
0 1 2 3 4 5 6 7 8 9
```

# Les tableaux 2D en paramètre

43

```
7  const int SIZE = 2;
8
9  // display 2D array with a nice formatted display
10 // array : the 2D array to display
11 void displayArray(int array[][SIZE], int size)
12 {
13     for (int i = 0; i < size; ++i) {
14         cout << " [";
15         for (int j = 0; j < size; ++j) {
16             cout << setw(3) << array[i][j];
17         }
18         cout << " ]" << endl;
19     }
20 }
```

la taille de la 2D  
doit être définie

```
112 int array2D[SIZE][SIZE] = {{11, 22}, {33, 44}};
113 displayArray(array2D, SIZE);
```

```
[ 11 22 ]
[ 33 44 ]
```

# Les tableaux en retour

44

```
30 // returns an array with each element equals to the index
31 // tries to return the array
32 int[] fillIntegerArray()                                ⚠ Function cannot return array type 'int[]'
33 { ⚠ Brackets are not allowed here; to declare an array, place the brackets after the name (fix available)
34     int array[10];
35     for(int index=0 ; index<10 ; index++)
36     {
37         array[index] = index;
38     }
39
40     return array;    ⚠ Cannot initialize return object of type 'int' with an lvalue of type 'int[10]'
41 }
```

Il n'est pas possible de retourner un tableau d'une fonction dans l'état actuel de nos connaissances

...

Nous verrons comment faire avec les pointeurs !

# Les enregistrements

45

```
5 struct Person
6 {
7     string name;
8     string firstname;
9     int birthYear;
10 };
11
12 // display Person data : "firstname, name, (birth year)."
13 // person : the Person (struct) data to display
14 void displayPerson(Person person)
15 {
16     cout << "Name : " << person.firstname << ", " << person.name << "(" << person.birthYear << ").";
17 }
18
19 // set Person birth year
20 // person : the Person (struct)
21 // birthyear : the Person birthyear
22 void setPersonBirtYear(Person person, int birthYear)
23 {
24     person.birthYear = birthYear;
25 }
26
27
94 Person boursorama{"Dujardin", "Jean", 1972};
95
96 displayPerson(boursorama) ;
97 cout << endl;
98 setPersonBirtYear(boursorama, 1980);
99 displayPerson(boursorama) ;
```

Name : Jean, Dujardin(1972).  
Name : Jean, Dujardin(1972).

# Les enregistrements

46

```
5 struct Person
6 {
7     string name;
8     string firstname;
9     int birthYear;
10 };
11
12 // display Person data : "firstname, name, (birth year)."
13 // person : the Person (struct) data to display
14 void displayPerson(Person person)
15 {
16     cout << "Name : " << person.firstname << ", " << person.name << "(" << person.birthYear << ").";
17 }
18
19 // set Person birth year
20 // person : the Person (struct)
21 // birthyear : the Person birthyear
22 void setPersonBirtYear(Person& person, int birthYear)
23 {
24     person.birthYear = birthYear;
25 }
26
27 Person boursorama{"Dujardin", "Jean", 1972};
28
29 displayPerson(boursorama) ;
30 cout << endl;
31 setPersonBirtYear(boursorama, 1980);
32 displayPerson(boursorama) ;
```

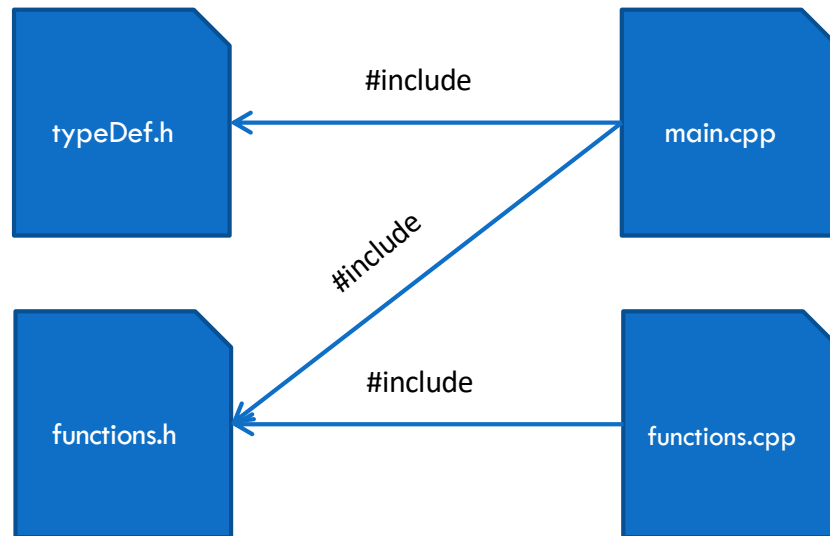
passage par référence  
explicite

Name : Jean, Dujardin(1972).  
Name : Jean, Dujardin(1980).



# Architecture d'un projet multi-fichiers

47



```
// commentaire en-tête du fichier de la main
#include <iostream>
using namespace std;

#include "typedef.h"
#include "functions.h"

int main() {

    ...

}
```

- **Compilation séparée**
- **Fichiers d'en-tête :**
  - **typedef.h** : déclaration des constantes, énumérations et enregistrements,
  - **functions.h** : déclaration des fonctions.
- **Fichiers sources**
  - **functions.cpp** : définition des fonctions,
  - **main.cpp** : programme principal (test ou release)

# DS1 3 exercices

48

	<b>ex1 /5</b>	<b>ex2 /7</b>	<b>ex3 /8</b>	<b>total /20</b>
<b>moyenne</b>	2,78	4,05	5,25	11,78
<b>min</b>	0	0	0	0
<b>max</b>	5	7	8	18,5

# DS1 - ex2 - calculateur de moyenne

49

## 2 Calculateur de moyenne (7 pts, 15')

Créez un programme qui lit une série de valeurs numériques réelles tapées successivement par l'utilisateur. Ces valeurs doivent être comprises entre 0 et 20. Si la valeur n'est pas acceptable, alors l'utilisateur voit un message qui lui demande de recommencer sa saisie de cette valeur jusqu'à ce qu'elle soit correcte.

La saisie de la valeur  $-1$  comme note signale au programme qu'il n'y a plus de note à taper ; cette note  $-1$  ne doit pas être considérée comme une vraie note pour la suite du problème, c'est uniquement une marque de fin.

A la fin du programme, la moyenne des valeurs saisies doit être affichée (Attention au cas où il n'y aurait pas de note...).