

Chapitre 5

SOUS-PROGRAMMES

POLYTECHNIQUE
MONTRÉAL

LE GÉNIE
EN PREMIÈRE CLASSE



Les fonctions

- ❑ Réduit la complexité des programmes.
- ❑ Ensemble d'instructions servant à exécuter une tâche particulière dans la résolution d'un problème.
- ❑ Chaque fonction peut être testée et mise au point indépendamment des autres.
- ❑ Un programme peut se résumer à une séquence d'appels à des fonctions.

Fonction

```
Type_resultat Nom(liste des types et paramètres)
{
    partie déclarations de la fonction < au besoin >

    instructions de la fonction

    return expression_de_retour; < au besoin >
}
```

Fonction

```

// Ordre d'exécution:
//
// 4
// 5 (cette ligne est exécutée 10 fois)
// 6 La fin de la fonction est atteinte ->
//   retourne où la fonction à été appelée.
//
void afficherNombresImpairs()
{
    for (int i = 11; i < 30; i += 2)
        cout << setw(4) << i;
}

int main()
{
    cout << "Nombres impairs de 10 à 30";
    cout << endl << endl;
    afficherNombresImpairs();
    cout << endl << endl;
    cout << "Affichage terminé";
}

```

est possible de retourner un code d'erreur au programme qui a appelé notre programme; par défaut le code est zéro, signifiant "pas d'erreur".

Début de l'exécution

Fonction: Les paramètres

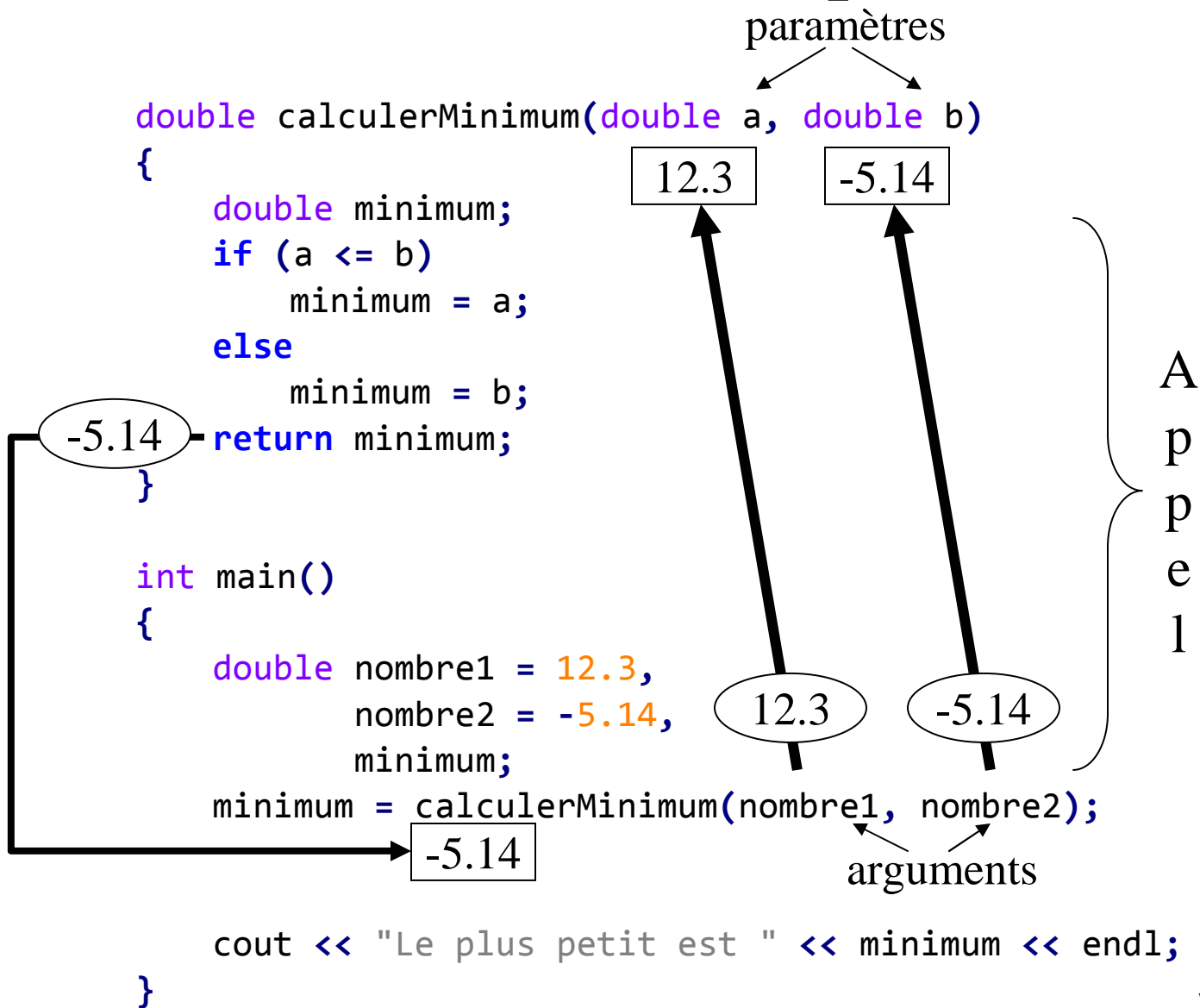
```
double calculerMinimum(double a, double b)
{
    double minimum;
    if (a <= b)
        minimum = a;
    else
        minimum = b;
    return minimum;
}

int main()
{
    double nombre1 = 12.3,
           nombre2 = -5.14,
           minimum;
    minimum = calculerMinimum(nombre1, nombre2);

    cout << "Le plus petit est " << minimum << endl;
}
```

V_parametres.cpp

Fonction: Les paramètres



V_parametres.cpp

Fonction: les paramètres

- ❑ Les paramètres servent à augmenter l'utilisation de la fonction en obtenant de l'information de l'extérieur lors de l'appel.
- ❑ Il doit y avoir le même nombre d'arguments lors de l'appel que de paramètres définis dans la fonction.
- ❑ L'ordre des arguments dans l'appel de la fonction est important puisque c'est la position de l'argument à l'appel qui détermine le paramètre associé.

Transmission de paramètres

PAR VALEUR

- La valeur de l'argument est transmise au paramètre correspondant.
- Le paramètre est initialisé à la valeur reçue en effectuant une copie de la valeur.
- Lorsque le paramètre est modifié, l'argument n'est pas modifié.

PAR ADRESSE

- L'adresse de l'argument est transmise au paramètre correspondant.
- Le paramètre est initialisé à l'adresse (de l'argument) reçue, et par le fait même réfère à la valeur de l'argument.
- Lorsque le paramètre est modifié, l'argument est modifié.

Transmission par valeur

```
void permuter(int a, int b)    // Passage par valeur.
```

```
{
```

```
    int tampon = a;
```

```
    a = b;
```

```
    b = tampon;
```

```
}
```

```
int main()
```

```
{
```

```
    int i = 5;
```

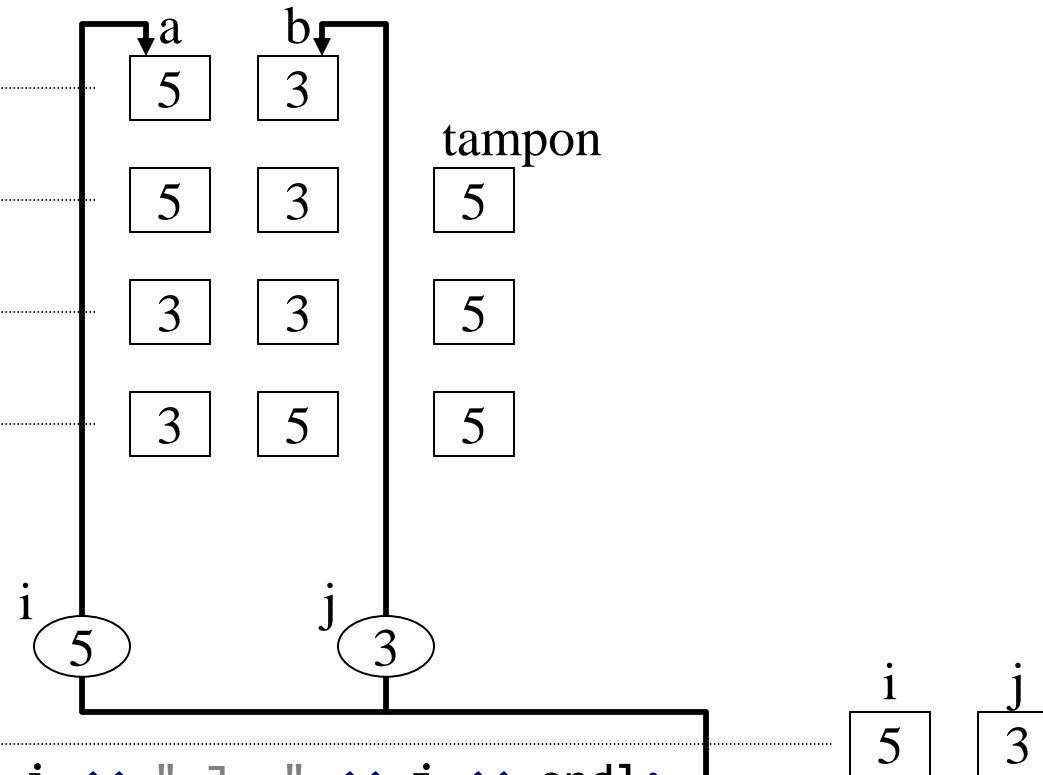
```
    int j = 3;
```

```
    cout << "I= " << i << " J= " << j << endl;
```

```
    permuter(i, j);
```

```
    cout << "I= " << i << " J= " << j << endl;
```

```
}
```



V_passage.cpp

Transmission par adresse

```
void permuter(int& a, int& b) // Passage par adresse.
```

```
{
```

```
    int tampon = a;
```

```
    a = b;
```

```
    b = tampon;
```

```
}
```

```
int main()
```

```
{
```

```
    int i = 5;
```

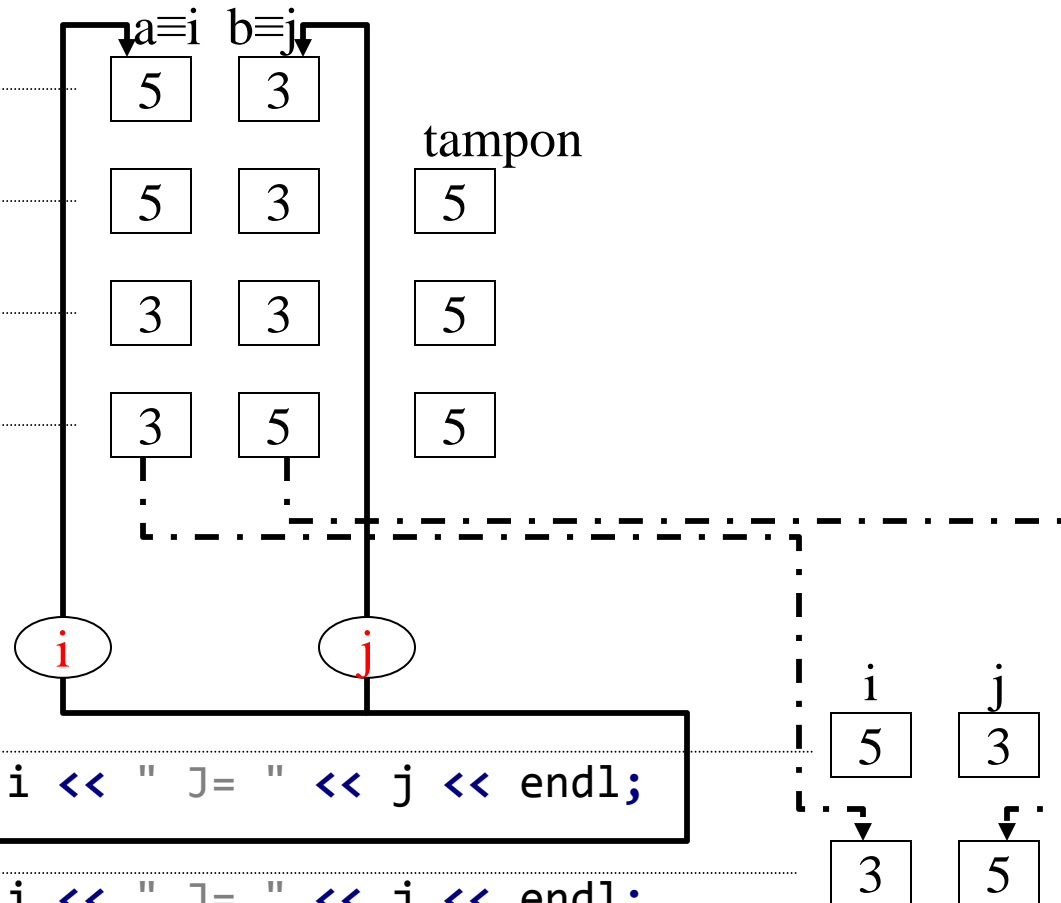
```
    int j = 3;
```

```
    cout << "I= " << i << " J= " << j << endl;
```

```
    permuter(i, j);
```

```
    cout << "I= " << i << " J= " << j << endl;
```

```
}
```



V_passage.cpp

Transmission par valeur et par adresse

```
#include <iostream>
#include <iomanip>
using namespace std;

void modifier(int entier1, int& entier2)
{
    entier1 = entier1 + 100;
    entier2 = entier2 + 100;
    cout << "(2) Les valeurs sont ";
    cout << setw(4) << entier1 << setw(4) << entier2 << endl;
}

int main()
{
    int compteur = 7, indice = 12;

    cout << "(1) Les valeurs sont ";
    cout << setw(4) << compteur << setw(4) << indice << endl;

    modifier(compteur, indice);

    cout << "(3) Les valeurs sont ";
    cout << setw(4) << compteur << setw(4) << indice << endl;
}
```

Résultat de l'exécution:

```
(1) Les valeurs sont      7  12
(2) Les valeurs sont    107 112
(3) Les valeurs sont      7 112
```

V_passage_valeur_adresse.cpp

Transmission de paramètres

- ❑ La transmission par valeur sert à fournir à la fonction une valeur dont elle a besoin pour exécuter correctement ses instructions. On qualifie le paramètre (IN).
- ❑ La transmission par adresse sert à retourner une valeur au point d'appel via l'argument. Dans ce cas on qualifie le paramètre (OUT). La fonction peut, au besoin, utiliser une valeur initiale. Dans ce cas on qualifie le paramètre (IN/OUT).
- ❑ La transmission par adresse peut aussi servir à éviter une copie de valeur. On qualifie alors le paramètre de (IN), et ajoutons « const » devant le type.

Transmission de paramètres

- ❑ Il doit y avoir correspondance entre les arguments de l'appel et les paramètres de la fonction pour ce qui est de leur nombre et leur type.
- ❑ L'argument dans un appel doit absolument être une variable si le paramètre correspondant est transmis par adresse. Il ne peut être une constante ni une expression.
- ❑ Si le paramètre est transmis par valeur, l'argument dans l'appel peut être une variable, une constante ou une expression.

**I
M
P
O
R
T
A
N
T**

Un tableau comme paramètre

- ❑ Toujours transmis par adresse (sans &) puisque l'identificateur d'un tableau contient comme information l'adresse du premier élément du tableau.

- void afficher_1D(int tableau[], int dim)
- void afficher_2D(int tableau[][5], int dim)
- void afficher_3D(int tableau[][10][10], int dim)
- void somme(int result[10], const int a[10], const int b[10])

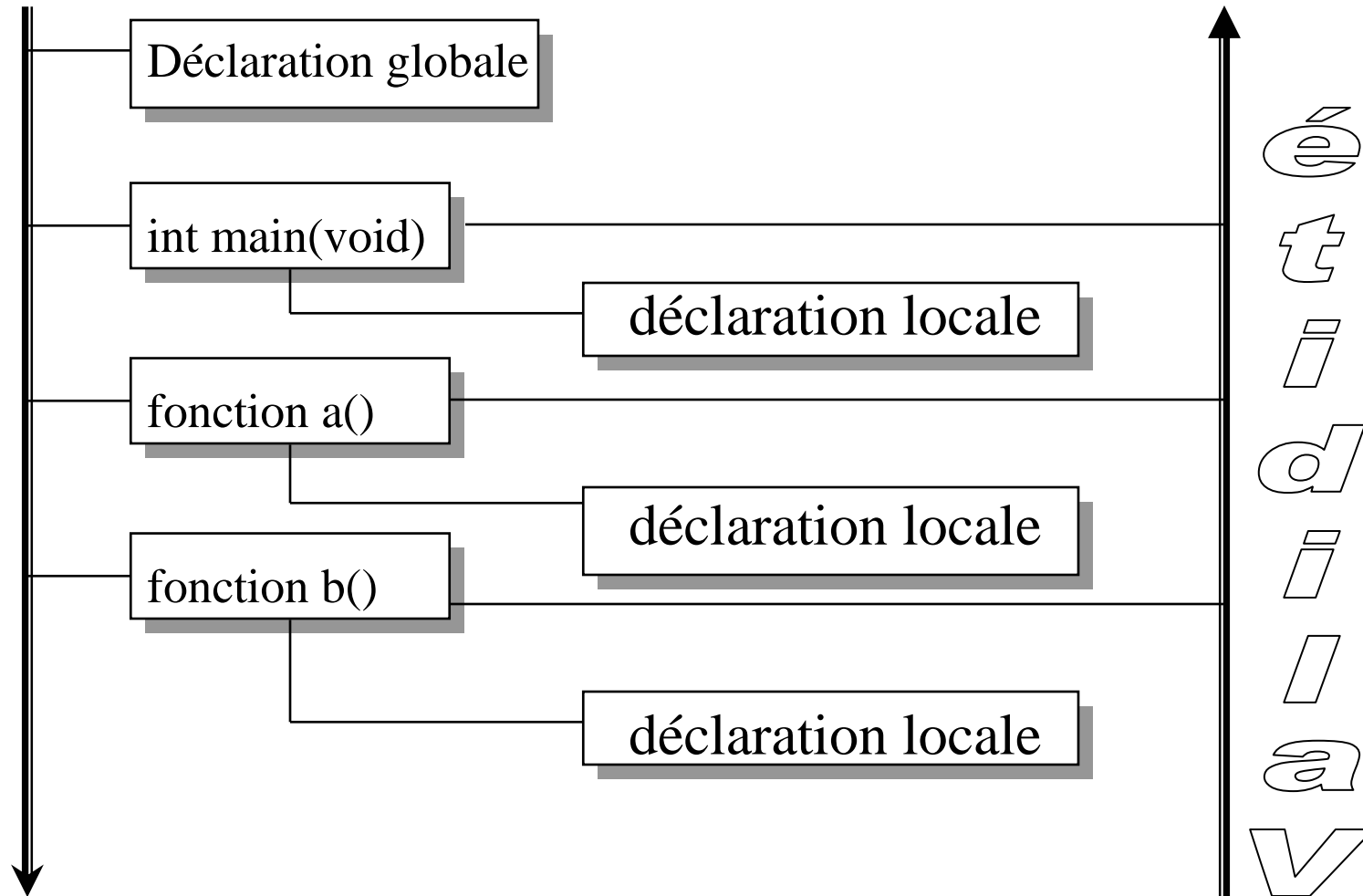
Ces dimensions doivent être précisées

Le tableau ne peut être modifié

Exercice

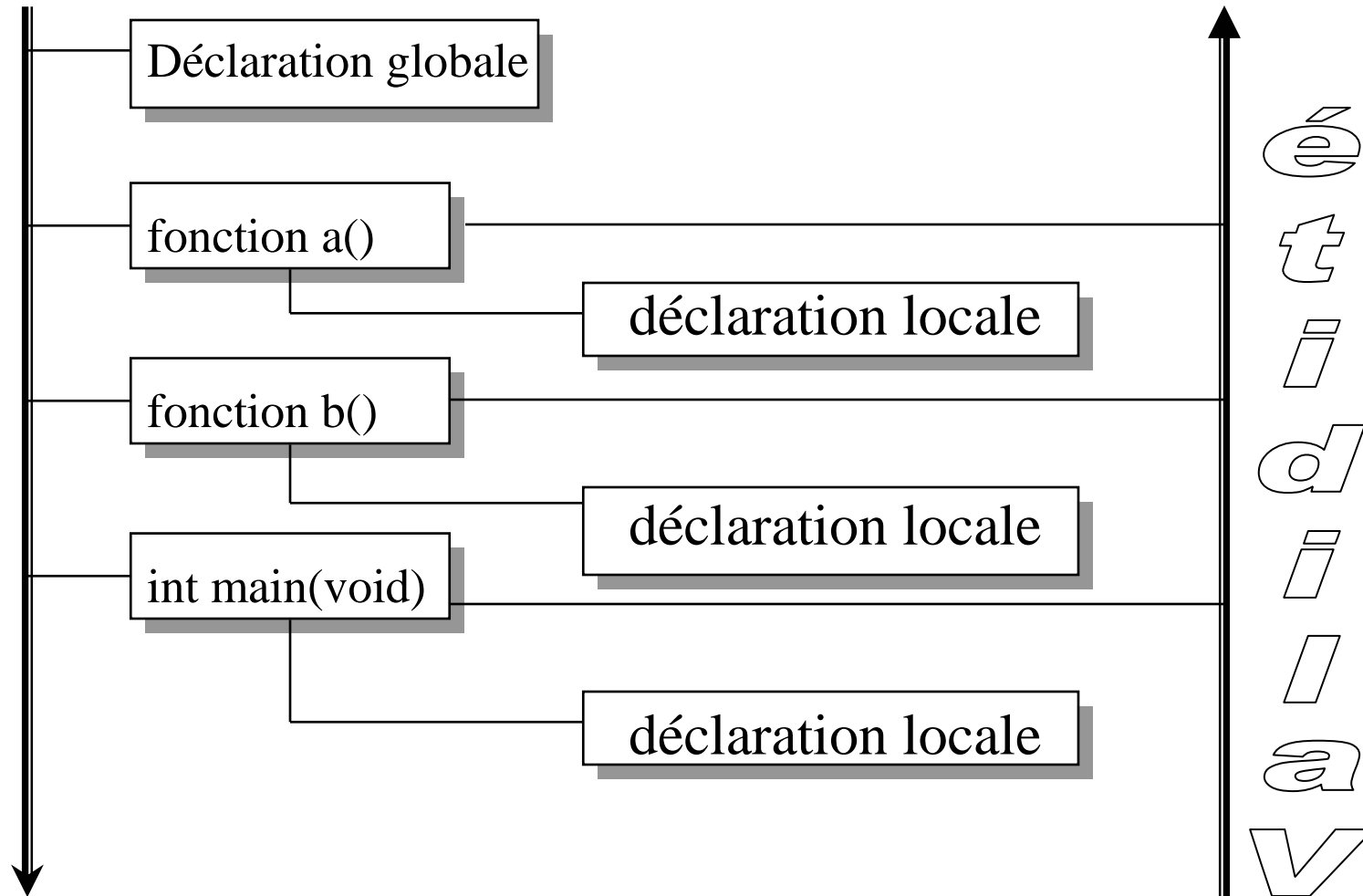
- Écrire un programme qui calcule la note obtenue lors d'une compétition. Ce programme contient:
 - une fonction qui lit la note des 8 juges;
 - une fonction qui calcule la note en les additionnant sauf la note maximale et la note minimale;
 - une fonction qui trouve la note min et la note max.

Domaine de validité



La fonction main() n'a accès à aucune fonction ==> prototypage

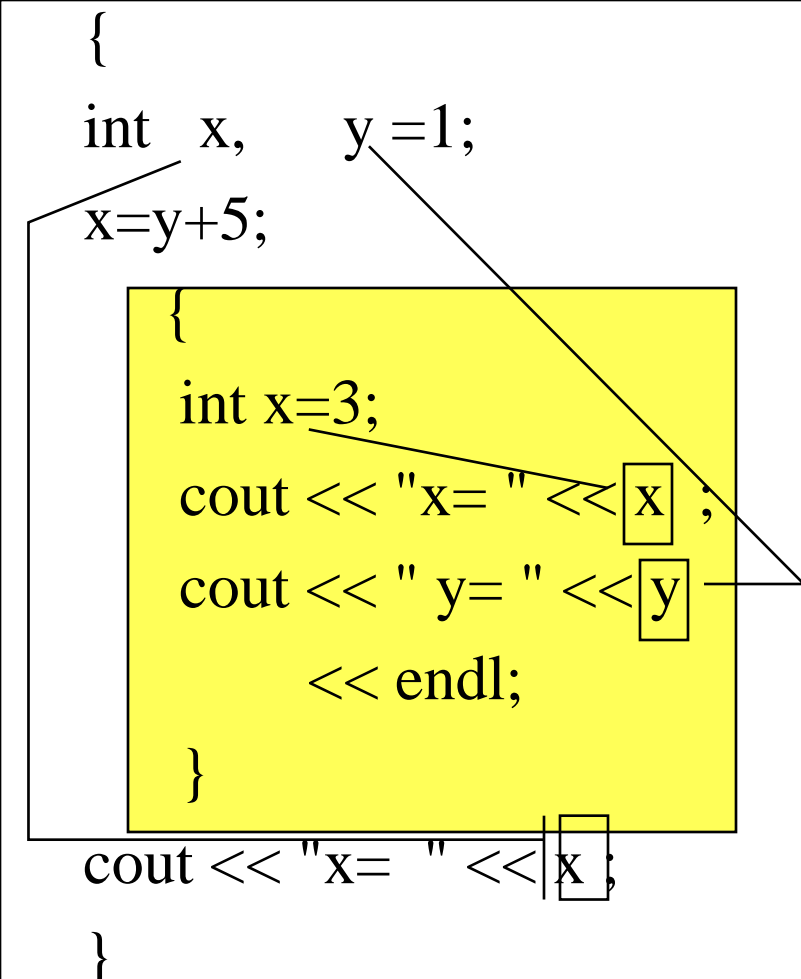
Domaine de validité



La fonction main() a accès à toutes les fonctions déclarées auparavant.

Domaine de validité

```
{  
  int x, y=1;  
  x=y+5;  
  {  
    int x=3;  
    cout << "x= " << x ;  
    cout << " y= " << y  
      << endl;  
  }  
  cout << "x= " << x ;  
}
```



Affichage

x= 3 y= 1

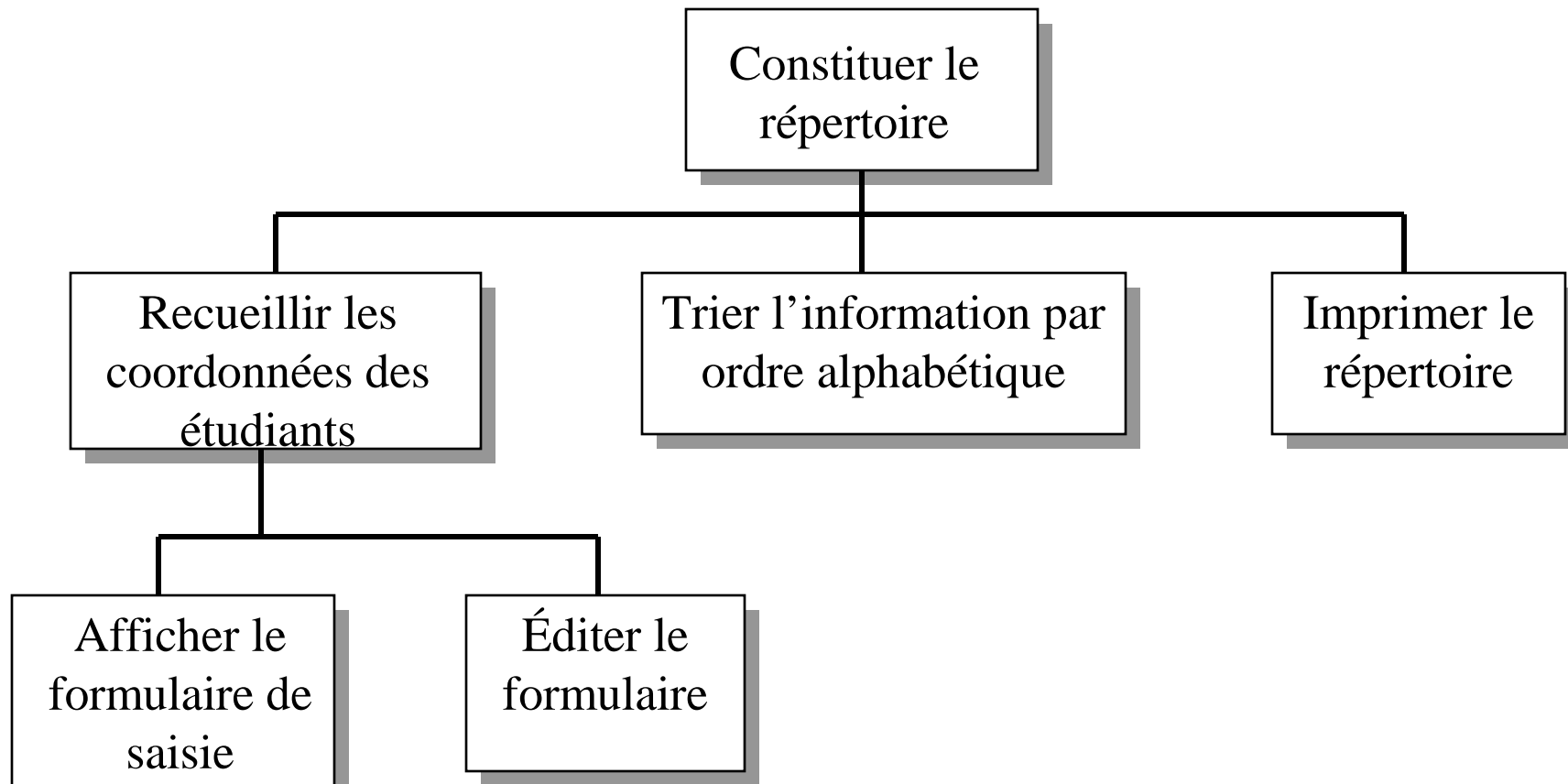
x= 6

Fonction

- ❑ void spécifie qu'il n'y a pas de valeur de retour
- ❑ Mode de transmission de paramètres
 - ⇒ Transmission par valeur
 - ⇒ Transmission par adresse à l'aide de référence [C++]
 - ⇒ Transmission par adresse à l'aide de pointeur [C]
- ❑ Initialisation des paramètres (Notion utilisée au chap.10)
- ❑ Surcharge de fonction (Notion utilisée au chap.10)
- ❑ Fonction récursive (n'est pas au programme du cours)

Algorithme

✂ Décomposition d'une tâche en sous-tâche



Algorithme

- ❑ Il faut rédiger un algorithme par fonction.
- ❑ Une opération correspondant à la tâche d'une fonction est précisée en ajoutant le nom de la fonction entre parenthèses.

Trier l'information par ordre alphabétique (Fonction trier())

Imprimer le répertoire téléphonique (Fonction imprimer())

Emplacement des fonctions

```
#include <iostream>
#include <iomanip>
using namespace std;
```

```
// Computes the total cost, including 5% sales tax,
// on nItems items at a cost of itemPrice each.
double totalCost(int nItems, double itemPrice);
```

Déclaration de fonction
ou prototype
(doit être avant l'utilisation)

```
int main()
{
    double pricePerItem;
    int numberOfItems;
    cout << "Enter the number of items purchased: ";
    cin >> numberOfItems;
    cout << "Enter the price per item $";
    cin >> pricePerItem;
```

```
    double bill = totalCost(numberOfItems, pricePerItem);
```

Appel de fonction

```
    cout << fixed << showpoint << setprecision(2);
    cout << numberOfItems << " items at "
        << "$" << pricePerItem << " each." << endl
        << "Final bill, including tax, is $" << bill
        << endl;
}
```

Entête de fonction

```
double totalCost(int nItems, double itemPrice)
{
    static const double TAXRATE = 0.05; // 5% sales tax
    double subtotal = itemPrice * nItems;
    return subtotal + subtotal*TAXRATE;
}
```

Corps de la
fonction

Définition de la fonction
(peut être après l'utilisation)

V_emplacement_apres.cpp

```
#include <iostream>
#include <iomanip>
using namespace std;
```

```
// Computes the total cost, including 5% sales tax,
// on nItems items at a cost of itemPrice each.
```

```
double totalCost(int nItems, double itemPrice)
```

Entête de fonction

```
{
    static const double TAXRATE = 0.05; // 5% sales tax
    double subtotal = itemPrice * nItems;
    return subtotal + subtotal*TAXRATE;
}
```

Corps de la
fonction

Définition de la fonction
(doit être avant l'utilisation
si pas de prototype)

```
int main()
```

```
{
    double pricePerItem;
    int numberOfItems;
    cout << "Enter the number of items purchased: ";
    cin >> numberOfItems;
    cout << "Enter the price per item $";
    cin >> pricePerItem;
```

```
double bill = totalCost(numberOfItems, pricePerItem);
```

Appel de fonction

```
cout << fixed << showpoint << setprecision(2);
cout << numberOfItems << " items at "
    << "$" << pricePerItem << " each." << endl
    << "Final bill, including tax, is $" << bill
    << endl;
```

```
}
```

V_emplacement_avant.cpp

Règles de portée (visibilité)

- Les fonctions sont des unités.
- Les variables des fonctions sont distinctes des autres variables déclarées à l'extérieur de la définition de la fonction.
- Des variables ayant le même identificateur dans différentes fonctions, ou différents blocs d'une fonction, sont distinctes.

Variables locales

```
double moyenne(double tableau[], int nombre);  
int main()  
{ double max, min, moy;  
  double liste[20];  
  int nombreElements;  
.....  
  moy = moyenne(liste, nombreElements);  
.....  
}
```

Variables locales
au main

```
double moyenne(double tableau[], int nombre)  
{ int i;  
  double moy;  
.....  
}
```

Variables locales à la
fonction moyenne

Il n'y a aucun lien entre les deux variables qui ont le nom 'moy'.

Variables locales (suite)

```
void main()  
{  
    .....  
    for (int i = 0; i < 10; i++)  
        for (int j = 0; j < 10; j++) {  
            .....  
        }  
    while (!trouver) {  
        int i, temp;  
        .....  
    }  
}
```

The diagram illustrates the scope of variables in the provided C code. Two callouts are present: one pointing to the nested 'for' loops with the text 'i, j variables locales aux blocs for', and another pointing to the 'while' loop with the text 'i, temp variables locales au bloc while'. The callouts are green ovals with lines pointing to the respective code blocks.

Il n'y a aucun lien entre les deux variables qui ont le nom 'i'.

Variables globales

- Une variable globale est accessible par toutes les fonctions définies dans le fichier y compris le main.

Variables Globales

```
double taux;
```

Variable globale

```
double moyenne( double tableau[], int nombre)
```

```
{ int i;
```

```
  double moy;
```

```
..... // utilisation de la variable taux
```

```
}
```

```
int main()
```

```
{ double max, min, moy;
```

```
  double liste[20];
```

```
  int nombreElements;
```

```
..... // utilisation de la variable taux
```

```
  moy = moyenne(liste, nombreElements);
```

```
.....
```

```
}
```

Éviter d'utiliser les
variables globales
dans le cours
INF1005C
(guide de codage point 48)

Les constantes globales sont tout à fait permises.

« Variables constantes »

```
void main()  
{  
    static const double pi = 3.1416;  
  
}
```

Constante locale

Les constantes « ne peuvent pas être modifiées » après leur déclaration.
Par contre:

```
void f(int x)  
{  
    const int y = x + 1;  
    static const int z = x + 1;  
    ...  
}
```

'y' a une valeur différente à chaque appel

'z' a la même valeur pour tous les appels;
le premier appel lui fixe sa valeur

Les variables les plus constantes sont donc 'static const'.