Introduction à la programmation en C++

Variables, structures conditionnelles, boucles et fonctions

Nicolas Audebert

21 septembre 2018

ONERA
THE FRENCH AEROSPACE LAB

École des Ponts
ParisTech

Plan de la séance

Dans l'épisode précédent...

Les langages de programmation

Les langages de programmation comportent une syntaxe et une grammaire permettant d'exprimer des instructions humainement compréhensibles, qui seront traduites pour être exécutées par la machine. Dans ce cours, nous étudierons le **C++**.

Compilation ou interprétation

C++ est un langage **compilé** : la traduction C++ $\rightarrow$ langage machine est réalisée par le **compilateur** avant l'exécution. L'exécutable généré peut être utilisé n'importe où.

Python est un langage **interprété** : la traduction Python $\rightarrow$ langage machine est réalisée à la volée par l'**interpréteur**, qui est nécessaire pour exécuter le programme.

Plan de la séance

Une brève histoire du C++

Logo du langage C++

Bjarne Stroustrup, créateur du C++ (Wikipédia)

Historique du C++

- 1983 : débuts du C++, premier compilateur
- 1985 : *The C++ Programming Language*, premier manuel de référence
- 1989 : version 2.0 du langage
- 1998 : première standardisation du langage
- 2003, 2011, 2014, 2017 : mises à jour du langage (actualisation du standard, nouvelles fonctionnalités)
- 2020 : prochaine standardisation

Pourquoi le C++ ?

C et C++

C++ est une augmentation du langage C pour lui en fournir de nouvelles fonctionnalités :

- Les classes (et la notion d'objets),
- La gestion des exceptions,
- La surcharge des opérateurs,
- Les templates,
- Les espaces de noms,
- ...

C++ comporte également une bibliothèque standard riche.

C++ est un langage **libre de droit**.

Plan de la séance

Type des variables

Langage typé

Dans un **langage typé**, les variables représentent une catégorie fixée d'objets et ne peuvent en changer. C++ est un langage typé.

```
1: int i;
2: i = 2;
3: float d = 5.6;
4: char a = 'z';
5: bool v = true;
```

1. Déclaration d'un entier **i**
2. Affectation de la valeur 2 à **i**
3. Déclaration et affectation d'un réel **d**
4. Idem pour un caractère **a**
5. Idem pour un booléen **v**

Type des variables

Langage typé

Dans un **langage typé**, les variables représentent une catégorie fixée d'objets et ne peuvent en changer. C++ est un langage typé.

```
1: int i;
2: i = 2;
3: float d = 5.6;
4: char a = 'z';
5: bool v = true;
```

1. Déclaration d'un entier **i**
2. Affectation de la valeur 2 à **i**
3. Déclaration et affectation d'un réel **d**
4. Idem pour un caractère **a**
5. Idem pour un booléen **v**

Type des variables

Langage typé

Dans un **langage typé**, les variables représentent une catégorie fixée d'objets et ne peuvent en changer. C++ est un langage typé.

```
1: int i;
2: i = 2;
3: float d = 5.6;
4: char a = 'z';
5: bool v = true;
```

1. Déclaration d'un entier `i`
2. Affectation de la valeur 2 à `i`
3. Déclaration et affectation d'un réel **d**
4. Idem pour un caractère `a`
5. Idem pour un booléen `v`

Type des variables

Langage typé

Dans un **langage typé**, les variables représentent une catégorie fixée d'objets et ne peuvent en changer. C++ est un langage typé.

```
1: int i;
2: i = 2;
3: float d = 5.6;
4: char a = 'z';
5: bool v = true;
```

1. Déclaration d'un entier `i`
2. Affectation de la valeur 2 à `i`
3. Déclaration et affectation d'un réel `d`
4. Idem pour un caractère `a`
5. Idem pour un booléen `v`

Type des variables

Langage typé

Dans un **langage typé**, les variables représentent une catégorie fixée d'objets et ne peuvent en changer. C++ est un langage typé.

```
1: int i;
2: i = 2;
3: float d = 5.6;
4: char a = 'z';
5: bool v = true;
```

1. Déclaration d'un entier **i**
2. Affectation de la valeur 2 à **i**
3. Déclaration et affectation d'un réel **d**
4. Idem pour un caractère **a**
5. Idem pour un booléen **v**

Déclaration et affectation d'une variable

Déclaration

La déclaration annonce l'existence d'une variable :

```
type nom_de_variable ;
```

Affectation

L'affectation assigne une valeur à une variable déclarée :

```
nom_de_variable = valeur ;
```

Il est possible de faire les deux en même temps :

```
type nom_de_variable = valeur;
```

```python
# Python
a = 2 # un entier
a = 3.4 # un réel
```

```cpp
// C++
a = 2; // ERREUR: a n'existe pas
int a; // Déclaration
a = 2; // Affectation
double pi = 3.14;
```

Les types natifs

C++ connaît un certain nombre de types numériques nativement.

Type	Espace	min	max
`int`	$\mathbb{Z}$	-2147483648	2147483647
`unsigned int`	$\mathbb{N}$	0	4294967295
`char`	$\mathbb{Z}$	-128	127
`unsigned char`	$\mathbb{Z}$	0	255
`float`	$\mathbb{R}$	$\pm 3,4e \pm 38$ ($\simeq$ 7 décimales)	
`double`	$\mathbb{R}$	$\pm 1,7e \pm 308$ ($\simeq$ 15 décimales)	
`bool`	$\{0, 1\}$	`false` (0)	`true` (1)

Exemple

```
int i = 2; // Définition et affectation
cout << i << endl; // Affiche "2"

int j;
j = i; // j est une nouvelle variable valant 2
i = 1; // Ne modifie que i, pas j !
cout << i << ", " << j << endl; // Affiche "1, 2"

int k, l, m = 4; // Définition multiple
k = l = 3;   // Affectation multiple
cout << k << ", " << l << ", " << m << endl; // Affiche "3, 3, 4"

int n = 5, o = n, p = INT_MAX; // Initialisations
cout << n << ", " << o << ", " << p << endl;
// Affiche 5, 5, 2 147 483 647

int q = r = 4; // Erreur ! (r n'existe pas)

const int s = 12; // Définit s comme constante valant 12
s = 13; // Erreur !
```

Conversions et règles de calcul

Les opérateurs mathématiques usuels sont définis et utilisables, mais attention aux **conversions implicites** (transtypage).

Opérateurs mathématiques

- **+** : addition
- **-** : soustraction
- ***** : multiplication
- **/** : division (entière ou non)
- **%** : modulo

```
int a = 15, b = 7;
int c = a/b; // c=2
int d = a % b; // d=1

double e = b/a; // ATTENTION e = 0
double f = double(b)/a; // f = 0.467
double g = (b+0.)/a // g = 0.467

float h = 5.6;
int i = h; // i = 5 ==> WARNING
int i = int(h); // i = 5, pas de
 ↪  warning
```

Exemples

```
int a = 4, b = 3;
double c = a/b;
// Que vaut c ?
```

Exemples

```
int a = 4, b = 3;
double c = a/b;
// Que vaut c ?

// c vaut 1, car la division a/b se fait en entier
```

Exemples

```
int a = 4, b = 3;
double c = a/b;
// Que vaut c ?

// c vaut 1, car la division a/b se fait en entier


double a = 4, b = 3;
int c = a/b;
// Que vaut c ?
```

Exemples

```cpp
int a = 4, b = 3;
double c = a/b;
// Que vaut c ?

// c vaut 1, car la division a/b se fait en entier


double a = 4, b = 3;
int c = a/b;
// Que vaut c ?

// c vaut 1. La division se fait en flottant (1,33) mais c est un int !
```

Exemples

```
int a = 4, b = 3;
double c = a/b;
// Que vaut c ?

// c vaut 1, car la division a/b se fait en entier


double a = 4, b = 3;
int c = a/b;
// Que vaut c ?

// c vaut 1. La division se fait en flottant (1,33) mais c est un int !


int a = 4;
double c = a/3.;
// Que vaut c ?
```

Exemples

```
int a = 4, b = 3;
double c = a/b;
// Que vaut c ?

// c vaut 1, car la division a/b se fait en entier


double a = 4, b = 3;
int c = a/b;
// Que vaut c ?

// c vaut 1. La division se fait en flottant (1,33) mais c est un int !


int a = 4;
double c = a/3.;
// Que vaut c ?

// c vaut 1,33, car 3. est un flottant, la division se fait donc en
 ↪  flottant.
```

Plan de la séance

Expressions booléennes

Une expression booléenne est une affirmation pouvant être soit **VRAIE** soit **FAUSSE**, qui peut dépendre des variables intervenant dans son écriture. On parle également de prédicat logique.

En C++

Le résultat d'une expression booléenne est un booléen, type `bool` qui prend les valeurs `true` ou `false`.

Opérateurs booléen et de comparaison

- **==** : égalité
- **!=** : différence
- **<** (**<=**) : infériorité
- **>** (**>=**) : supériorité
- **&&** : ET logique
- **||** : OU logique
- **!** : NON logique

```
int a = 3, b = 5, c == 8;
bool b1 = a == b; // b1 = 0 (false)
bool b2 = (a+c) != b // b2 = 1 (true)
cout << ( c >= a ) << endl; // 1
bool b3 = b2 && (a+b == c) // b3 = 1
cout << b3 || b1 << endl; // 1
cout << ! b3 << endl; // 0
```

Priorités

Le **ET** est prioritaire sur le **OU**

a XOR b

```
a && (!b) || (!a) && b
```

Opérateurs booléen et de comparaison

- **==** : égalité
- **!=** : différence
- **<** (**<=**) : infériorité
- **>** (**>=**) : supériorité
- **&&** : ET logique
- **||** : OU logique
- **!** : NON logique

```
int a = 3, b = 5, c == 8;
bool b1 = a == b; // b1 = 0 (false)
bool b2 = (a+c) != b // b2 = 1 (true)
cout << ( c >= a ) << endl; // 1
bool b3 = b2 && (a+b == c) // b3 = 1
cout << b3 || b1 << endl; // 1
cout << ! b3 << endl; // 0
```

Priorités

Le **ET** est prioritaire sur le **OU**

a XOR b

```
a && (!b) || (!a) && b
```

Opérateurs booléen et de comparaison

- **==** : égalité
- **!=** : différence
- **<** (**<=**) : infériorité
- **>** (**>=**) : supériorité
- **&&** : ET logique
- **||** : OU logique
- **!** : NON logique

```
int a = 3, b = 5, c == 8;
bool b1 = a == b; // b1 = 0 (false)
bool b2 = (a+c) != b // b2 = 1 (true)
cout << ( c >= a ) << endl; // 1
bool b3 = b2 && (a+b == c) // b3 = 1
cout << b3 || b1 << endl; // 1
cout << ! b3 << endl; // 0
```

Priorités

Le **ET** est prioritaire sur le **OU**

a XOR b

```
a && (!b) || (!a) && b
```

SI (...) FAIRE ...SINON ...

Syntaxe

`if(expression booléenne){...} else {...}`

```
int annee;
cout << "Entrez une année" << endl;
cin >> annee;

if(annee % 4 == 0){
    if(annee % 100 == 0 && annee % 400 != 0){
        cout << "Cette année n'est pas bissextile" << endl;
    } else {
        cout << "Année bissextile" << endl;
    }
} else {
    cout << "Cette année n'est pas bissextile" << endl;
}
```

Le switch

Le **switch** permet une énumération de ce qu'il faut faire en fonction des valeurs d'une variable **entière**.

```
char c;
if (c == 'a'){
    cout << "Lettre 'a'";
}
else if (c == 'f'){
    cout << "Lettre 'f'";
} else if(c=='e'||c=='i'||c=='o'
            ||c=='u'||c=='y') {
    cout << "Autre voyelle";
} else {
    cout << "Autre chose";
}
```

```
char c;
switch ( c ) {
case 'a' :
    cout << "Lettre 'a'";
    break;
case 'f' :
    cout << "Lettre 'f'";
    break;
case 'e' :
case 'i' :
case 'o' :
case 'u' :
case 'y' :
    cout<<"Autre voyelle";
    break;
default :
    cout<<"Une consonne";
    break;
```

Plan de la séance

Portée des variables

La règles des accolades

Une variable n'existe que dans le bloc (et les sous-blocs), défini par des accolades, dans lequel elle a été déclarée.

```python
a = 3
b = 6
if a < b:
    print(a)
    c = a + b
    print(c) # OK
print(c) # OK, c existe en
 ↪  dehors de son bloc
```

```cpp
int a = 3, b = 6;
if(a < b){
    cout << a << endl; // OK
    int c = a + b;
    cout << c << endl; // OK
}
cout << c << endl; // ERREUR !
```

Plan de la séance

TANT QUE (...) FAIRE ...

Syntaxe

```
while(expression booléenne){...}
do{...}while(expression booléenne);
```

```python
# Python
a = 30
while a>0:
    print(a)
    a -= 2
```

```cpp
// C++
int a = 30;
while(a > 0){
    cout << a << endl;
    a -=2;
}

int b = 1;
do {
    b *= 2;
}
while(b != 1024);
```

POUR (...) FAIRE ...

Syntaxe

```
for(initialisation;expression booléenne;itération)
{...}
```

```python
# Python
for a in range(10):
    print(a)

# C'est équivalent à :

a = 0
while a < 10:
    print(a)
    a +=1
```

```cpp
// C++
for(int a = 0; a < 10; a++){
// a++ signifie a = a + 1
    cout << a << endl;
}

# C'est équivalent à :

int a = 0;
while(a < 10){
    cout << a << endl;
    a++;
}
```

Sortir des boucles

L'instruction `break` permet de forcer la sortie d'une boucle.

```
/*
Sortir d'une boucle
 ↪  infinie en C++
*/
int a = 1;
while(true){ // boucle
 ↪  infinie
    a *= 2;
    cout << a << endl;
    if(a > 5000){
        break;
    }
}
```

```
/*
Sortir avant la fin d'une boucle for en
 ↪  C++
*/
for(int i = 0; i < 10; i++){
    int res;
    cout << "Entrez un entier" << endl;
    cin >> res;
    if(res == 5){
        cout << "5" << endl;
        break;
    }
}
```

Plan de la séance

Sens mathématique

En C++, le concept de fonction est le même qu'en mathématiques :

$$\begin{array}{rcccc} f & : & A & \to & B \\ & & x & \mapsto & f(x) \end{array}$$

Sens mathématique

En C++, le concept de fonction est le même qu'en mathématiques :

$$\begin{array}{rccc} f & : & A & \to & B \\ & & x & \mapsto & f(x) \end{array}$$

```
B f(A x){
...
return f(x);
}
```

Sens mathématique

En C++, le concept de fonction est le même qu'en mathématiques :

$$\begin{array}{rcl} f : A & \to & B \\ x & \mapsto & f(x) \end{array}$$

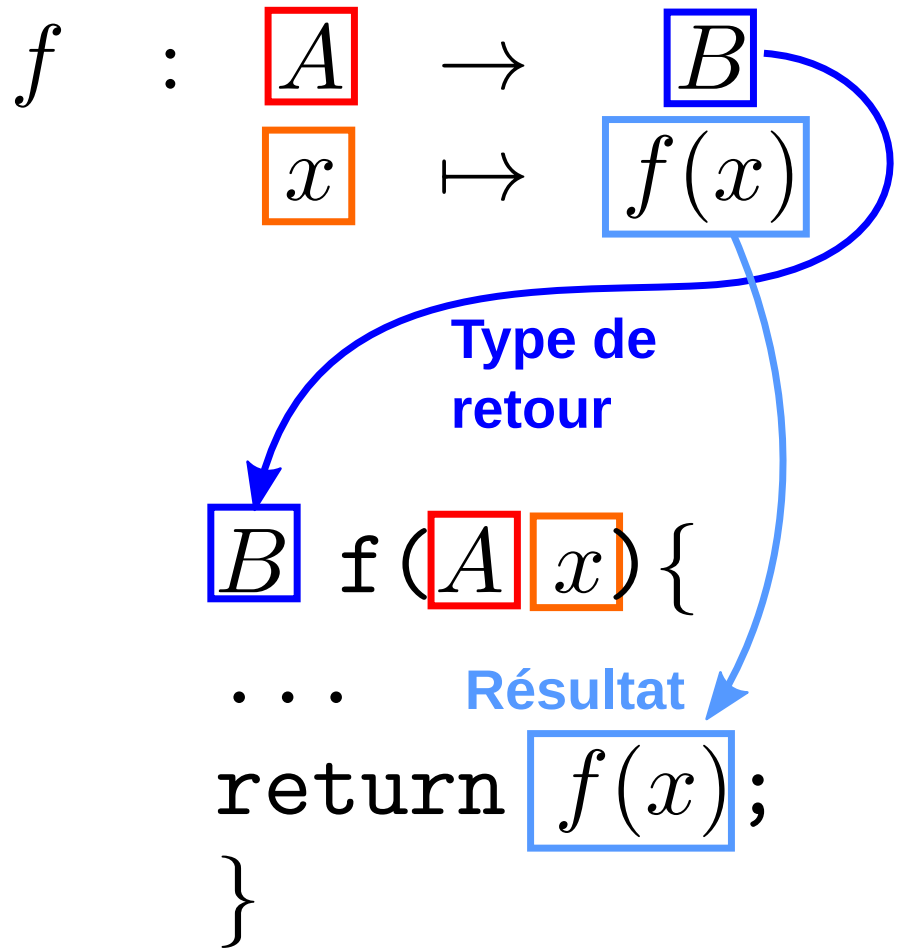
Type de l'argument

Nom de l'argument

```
B f(A x){
...
return f(x);
}
```

Sens mathématique

En C++, le concept de fonction est le même qu'en mathématiques :



Exemple

La fonction signe

$$\begin{array}{rccl} signe: & \mathbb{R} & \to & \{-1,0,1\} \\ & x & \mapsto & \begin{cases} -1 \text{ si } x < 0 \\ 1 \text{ si } x > 0 \\ 0 \text{ sinon} \end{cases} \end{array}$$

Deux manières d'écrire la même fonction.

```c
int signe(double x){
  int s = 0;

  if(x < 0)
    s = -1;
  if(x > 0)
    s = 1;
  return s;
}
```

```c
int signe(double x){
    if(x < 0)
        return -1;
    if(x > 0)
        return 1;
    return 0;
}
```

Fonction sans retour

Pour des fonctions qui ne renvoient rien, on utilise le type de retour vide : **`void`**.

Le retour vide : **`return;`** est optionnel.

Afficher

Afficher les coordonnées (x, y, z) d'un vecteur :

$$\mathit{affiche} : \quad \mathbb{R}^3 \quad \rightarrow \quad \emptyset$$

```
void affiche(double x, double y, double z){
    cout << "(" << x << "," << y << "," << z << ")" << endl;
    return; // OPTIONNEL
}
void affiche(double x, double y, double z){
    cout << "(" << x << "," << y << "," << z << ")" << endl;
}
```

Limitations

- Il n'est pas possible de renvoyer plusieurs valeurs.

```python
def f():
    a = 3
    b = 5
    return a, b # OK
```

```cpp
int f(){
    int a = 3, b = 5;
    return a, b; // ERREUR
}
```

- On ne peut pas modifier les arguments (ils sont copiés).

```cpp
void switch_1(double a,
              double b){
    // Échange a et b
    double c = b;
    b = a;
    a = c;
}
```

```cpp
int main(){
    double x = 5, y = 7;
    switch_1(x,y);
    cout << x << ", " << y;
    // Affiche "5, 7"
}
```

Le passage par référence

Ces problèmes sont dus à la copie qui est réalisée lorsque l'on entre dans une fonction.

Le passage par référence est une solution aux problèmes précédents. Il autorise la modification de l'argument concerné en le préfixant dans la signature de la fonction par &.

Syntaxe

```
type f(type1 & arg1, type2 & arg2, type3 arg3 ...){...}
```

Passage par référence : exemples

- ”Renvoyer” deux ou plus valeurs (modifier les arguments)

```
void f(int & a, int & b){
    a = 3;
    b = 5;
}
```

```
int main(){
    int x = 0, y = 0;
    f(x, y);
    cout << x << ", " << y;
    // Affiche "3, 5"
}
```

- Modifier les arguments

```
void switch_2(double & a,
              double & b){
    // Échange a et b
    double c = b;
    b = a;
    a = c;
}
```

```
int main(){
    double x = 5, y = 7;
    switch_2(x,y);
    cout << x << ", " << y;
    // affiche 7, 5
}
```

Passage par référence : zoom sur la mémoire 1

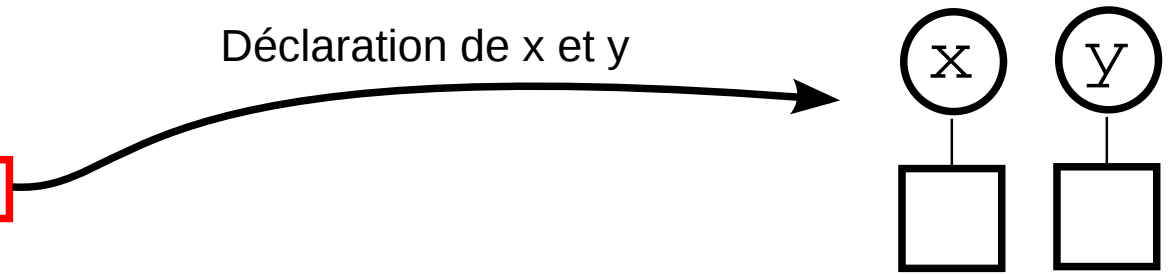
Fonctionnement de la fonction `switch_1`

```
int main(){
  ...
  double x,y;
  x = 5;
  y = 7;
  switch_1(x,y);
}
```

Passage par référence : zoom sur la mémoire 1

Fonctionnement de la fonction `switch_1`

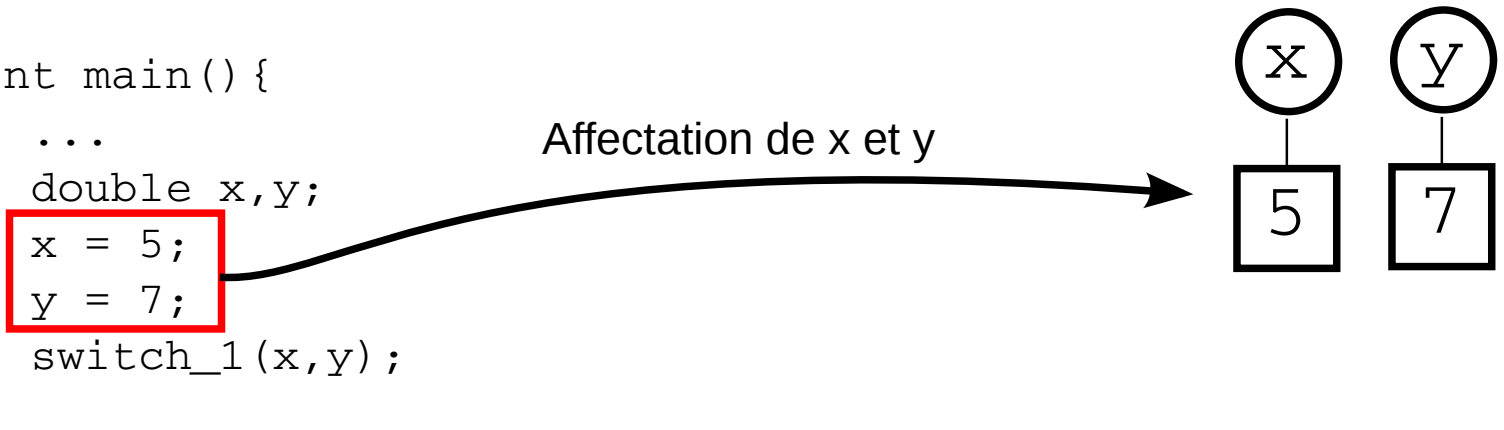
```
int main(){
  ...
  double x,y;
  x = 5;
  y = 7;
  switch_1(x,y);
}
```



Passage par référence : zoom sur la mémoire 1

Fonctionnement de la fonction `switch_1`

```
int main(){
  ...
  double x,y;
  x = 5;
  y = 7;
  switch_1(x,y);
}
```

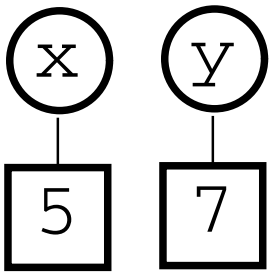


Passage par référence : zoom sur la mémoire 1

Fonctionnement de la fonction `switch_1`

```
int main(){
   ...
   double x,y;
   x = 5;
   y = 7;
   switch_1(x,y);
}
```

```
void switch_1(double a,
      double b){
   double c=b;
   b=a;
   a=c;
}
```

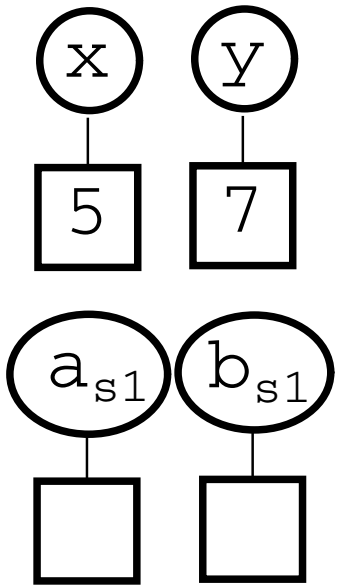


Passage par référence : zoom sur la mémoire 1

Fonctionnement de la fonction `switch_1`

```
int main(){
   ...
   double x,y;
   x = 5;
   y = 7;
   switch_1(x,y);
}
```

```
void switch_1(double a,
      double b){
   double c=b;
   b=a;
   a=c;
}
```

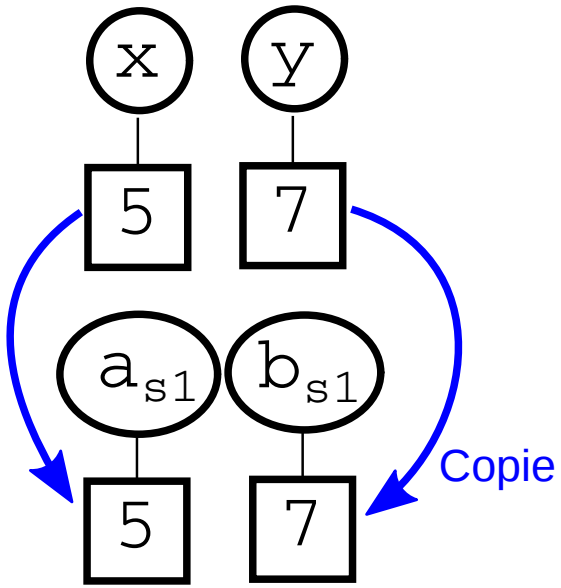


Passage par référence : zoom sur la mémoire 1

Fonctionnement de la fonction `switch_1`

```
int main(){
   ...
   double x,y;
   x = 5;
   y = 7;
   switch_1(x,y);
}
```

```
void switch_1(double a,
      double b){
   double c=b;
   b=a;
   a=c;
}
```

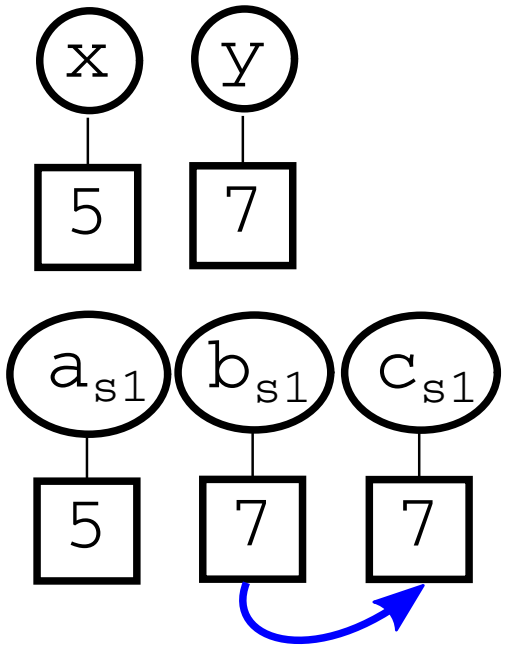


Passage par référence : zoom sur la mémoire 1

Fonctionnement de la fonction `switch_1`

```
int main(){
   ...
   double x,y;
   x = 5;
   y = 7;
   switch_1(x,y);
}
```

```
void switch_1(double a,
      double b){
   double c=b;
   b=a;
   a=c;
}
```

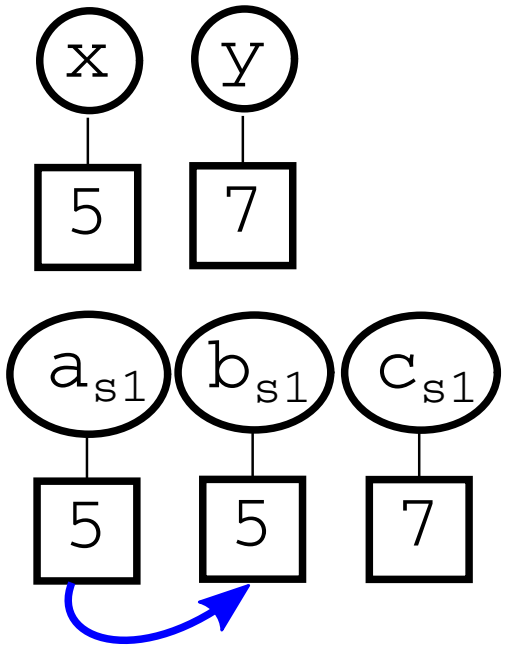


Passage par référence : zoom sur la mémoire 1

Fonctionnement de la fonction `switch_1`

```
int main(){
   ...
   double x,y;
   x = 5;
   y = 7;
   switch_1(x,y);
}
```

```
void switch_1(double a,
      double b){
   double c=b;
   b=a;
   a=c;
}
```

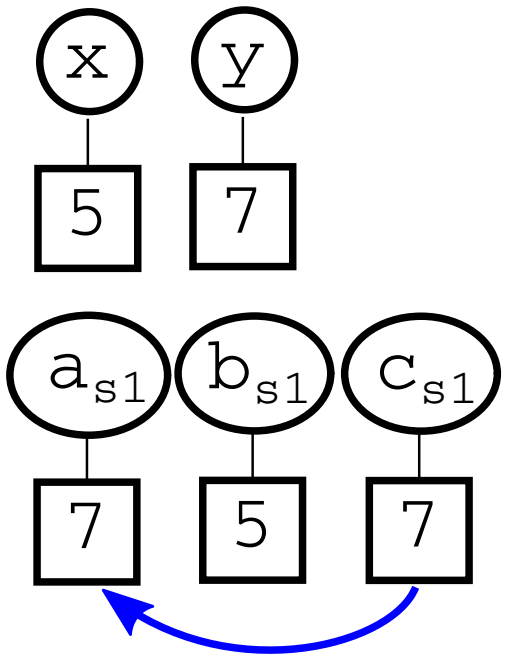


Passage par référence : zoom sur la mémoire 1

Fonctionnement de la fonction `switch_1`

```
int main(){
   ...
   double x,y;
   x = 5;
   y = 7;
   switch_1(x,y);
}
```

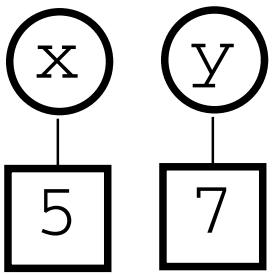
```
void switch_1(double a,
      double b){
   double c=b;
   b=a;
   a=c;
}
```



Passage par référence : zoom sur la mémoire 1

Fonctionnement de la fonction `switch_1`

```
int main(){
   ...
   double x,y;
   x = 5;
   y = 7;
   switch_1(x,y);
}
```



Passage par référence : zoom sur la mémoire 2

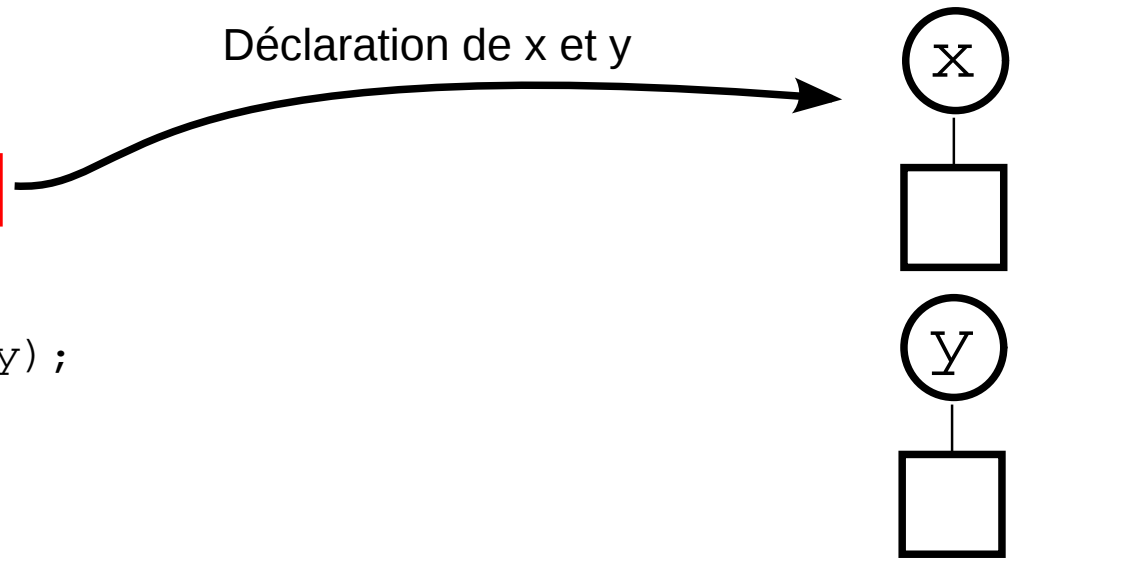
Fonctionnement de la fonction `switch_2`

```
int main(){
  ...
  double x,y;
  x = 5;
  y = 7;
  switch_1(x,y);
}
```

Passage par référence : zoom sur la mémoire 2

Fonctionnement de la fonction `switch_2`

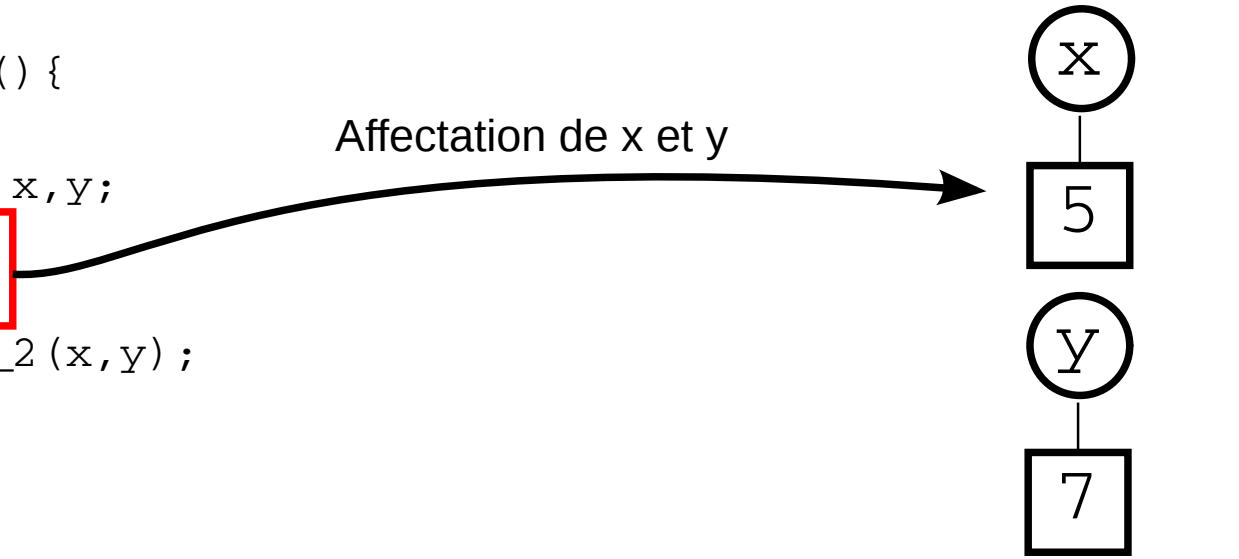
```
int main(){
  ...
  double x,y;
  x = 5;
  y = 7;
  switch_2(x,y);
}
```



Passage par référence : zoom sur la mémoire 2

Fonctionnement de la fonction `switch_2`

```
int main(){
   ...
   double x,y;
   x = 5;
   y = 7;
   switch_2(x,y);
}
```

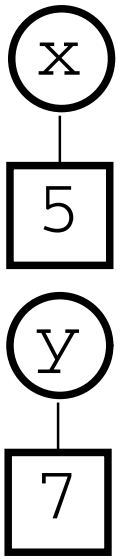


Passage par référence : zoom sur la mémoire 2

Fonctionnement de la fonction `switch_2`

```
int main(){
   ...
   double x,y;
   x = 5;
   y = 7;
   switch_2(x,y);
}
```

```
void switch_1(double & a,
      double & b){
   double c=b;
   b=a;
   a=c;
}
```

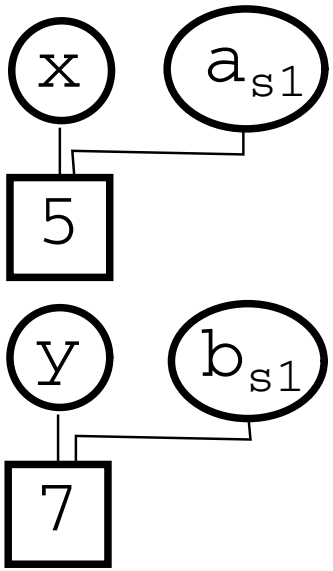


Passage par référence : zoom sur la mémoire 2

Fonctionnement de la fonction `switch_2`

```
int main(){
   ...
   double x,y;
   x = 5;
   y = 7;
   switch_2(x,y);
}
```

```
void switch_1(double & a,
     double & b){
   double c=b;
   b=a;
   a=c;
}
```



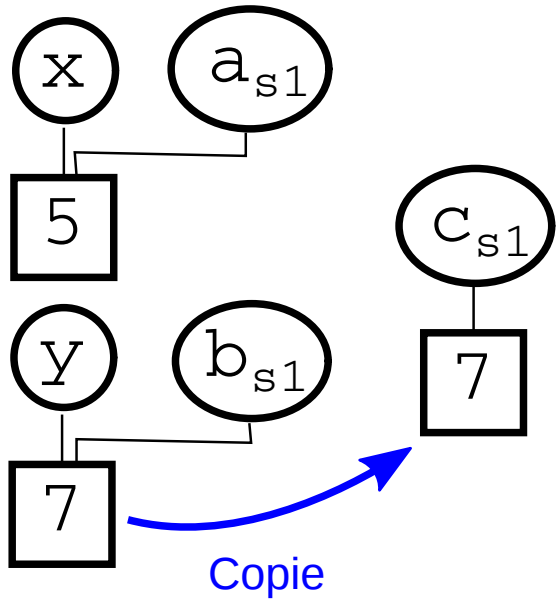
Partage de la mémoire

Passage par référence : zoom sur la mémoire 2

Fonctionnement de la fonction `switch_2`

```
int main(){
   ...
   double x,y;
   x = 5;
   y = 7;
   switch_2(x,y);
}
```

```
void switch_1(double & a,
      double & b){
   double c=b;
   b=a;
   a=c;
}
```

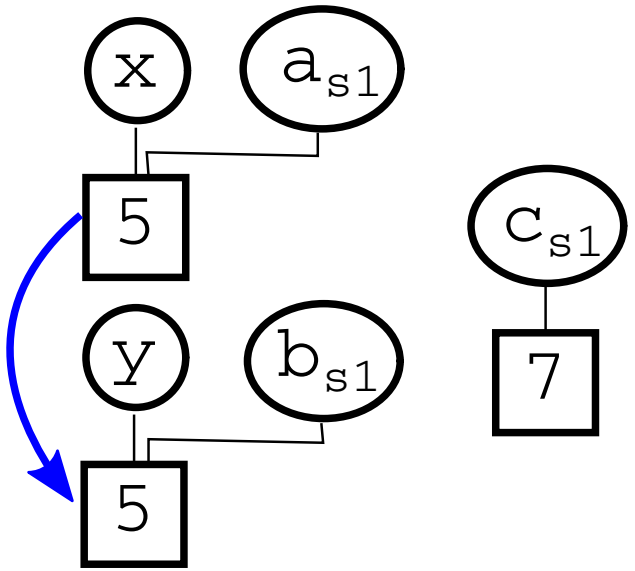


Passage par référence : zoom sur la mémoire 2

Fonctionnement de la fonction `switch_2`

```
int main(){
   ...
   double x,y;
   x = 5;
   y = 7;
   switch_2(x,y);
}
```

```
void switch_1(double & a,
      double & b){
   double c=b;
   b=a;
   a=c;
}
```

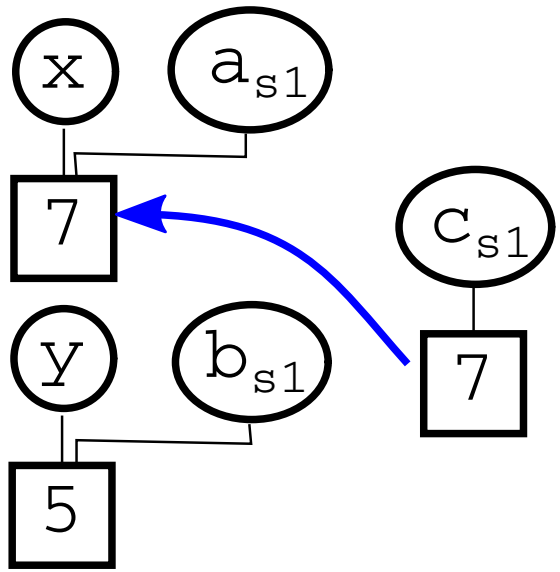


Passage par référence : zoom sur la mémoire 2

Fonctionnement de la fonction `switch_2`

```
int main(){
   ...
   double x,y;
   x = 5;
   y = 7;
   switch_2(x,y);
}
```

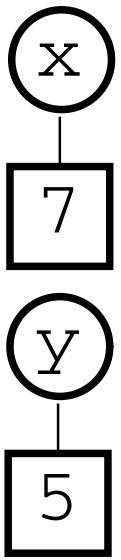
```
void switch_1(double & a,
      double & b){
   double c=b;
   b=a;
   a=c;
}
```



Passage par référence : zoom sur la mémoire 2

Fonctionnement de la fonction `switch_2`

```
int main(){
   ...
   double x,y;
   x = 5;
   y = 7;
   switch_2(x,y);
}
```



Exemples

```
int f(double a, int &b){
  a = 2.5;
  b = 1;
  return a/b;
}
double a = 3.14;
int b = 2;
int c = f(a,b);
// Que valent a, b et c ?
```

Exemples

```
int f(double a, int &b){
  a = 2.5;
  b = 1;
  return a/b;
}
double a = 3.14;
int b = 2;
int c = f(a,b);
// Que valent a, b et c ?

// a = 3.14 (pas modifié), b = 1 (référence), c = 2 car c'est un int !
```

Exemples

```
int f(double a, int &b){
  a = 2.5;
  b = 1;
  return a/b;
}
double a = 3.14;
int b = 2;
int c = f(a,b);
// Que valent a, b et c ?

// a = 3.14 (pas modifié), b = 1 (référence), c = 2 car c'est un int !

double g(double &a, double &b){
  return (a + b)/2;
}
double a = 9, b = 0;
double c = g(a,b);
// Que valent a, b et c ?
```

Exemples

```
int f(double a, int &b){
  a = 2.5;
  b = 1;
  return a/b;
}
double a = 3.14;
int b = 2;
int c = f(a,b);
// Que valent a, b et c ?

// a = 3.14 (pas modifié), b = 1 (référence), c = 2 car c'est un int !

double g(double &a, double &b){
  return (a + b)/2;
}
double a = 9, b = 0;
double c = g(a,b);
// Que valent a, b et c ?

// a=9 et b=0 sont passés par référence mais non modifiés, c=4.5
```

Surcharge

Il est possible de donner le même nom à deux fonctions (ou plus) à condition que les types ou le nombre des arguments de celles-ci soient différents. Autrement, deux fonctions peuvent avoir le même nom si leurs signatures diffèrent.

```
int f(int v1, int v2){
    return v1 + v2;
}

double f(int v1, int v2){
    return 0.5;
}

// ERREUR !!!
// Même nom +
// arguments identiques
```

```
int f(int v1, int v2){
    return v1 + v2;
}

double f(int v1, int v2, int
 ↪   v3){
    return 0.5;
}

// OK
// Même nom mais
// arguments différents
```

Portée, déclaration

Appel de fonction

Comme pour les variables on ne peut utilisée une fonction que si elle a été déclarée préalablement.

```
void f(){
    g(3);
    // ERREUR g est inconnue
}

void g(int i){
    ...
}
```

```
void g(int i){
    ...
}

void f(){
    g(3); //OK
}
```

Portée, déclaration

```
void f(){
    g(3);
    // ERREUR g est inconnue
}

void g(int i){
    f();
}
```

```
void g(int i); // Déclaration

void f(){
    g(3); // OK

// Définition
void g(int i){
    ...
}
```

Variables locales et globales

Variables globales

Les variables globales sont déclarées en dehors des fonctions. Elles sont accessibles à l'ensemble du programme.

```
void f(){
    ...
    int i1 = 3;
    ...
}

void g(int i){
    cout << i1 << endl;
    /* ERREUR i1 inconnu
       i1 est une variable
       de f */
}
```

```
int i1; // variable globale

void f(){
    ...
    i1 = 3;
    ...
}

void g(int i){
    cout << i1 << endl;
    /* OK i1 est une variable
       globale */
}
```

Variables locales et globales

Attention

L'usage des variables globales est à limiter au maximum :

- La communication entre les fonctions est source de bugs.
- Elles rendent les fonctions difficilement réutilisables.

Usage

Les variables globales sont des constantes générales.

```
const int width = 800; // constante non modifiable
const int height = 600;

int main(){
    ...
    openWindow(width, height);
    ...
    height = 5; // ERREUR : height est une constante
}
```

Plan de la séance

Organisation des TPs

- En salle informatique de 9h45 à 11h15.
- Par groupe de 2
- Sur machine de l'école ou machine perso
- OS au choix
  - Machine perso : Linux, Windows, OSX
  - École : Windows, Linux
- IDE au choix (QtCreator recommandé)
- À rendre sous forme d'archive `.zip` sur **Educnet** avant le **26/09**
- **Exercice individuel** à rendre pour le **3/10**

Organisation des TPs

- En salle informatique de 9h45 à 11h15.
- Par groupe de 2
- Sur machine de l'école ou machine perso
- OS au choix
  - Machine perso : Linux, Windows, OSX
  - École : Windows, Linux
- IDE au choix (QtCreator recommandé)
- À rendre sous forme d'archive `.zip` sur **Educnet** avant le **26/09**
- **Exercice individuel** à rendre pour le **3/10**

TP de la semaine

Sujet

- Utilisation du débogueur
- Prise en main de la bibliothèque Imagine++
- Programmation d'un mini-jeu de tennis