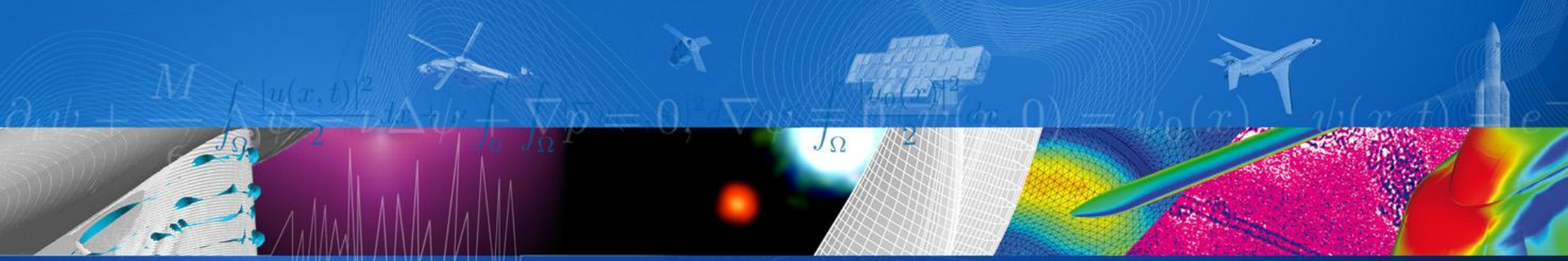


Chap. 3 : Variables, Tests, Boucles et Fonctions

Introduction au C++

Nicolas Audebert

ONERA
THE FRENCH AEROSPACE LAB

retour sur innovation

Plan de la séance

Le C++ un langage typé

Langage typé

Dans un langage typé, les variables appartiennent à un espace donné et ne peuvent en changer.

C++

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

- 1, déclaration d'un entier `i` puis affectation de la valeur 2 (2)
- 3, déclaration et affectation d'un réel `i`
- 4, idem pour un caractère
- 5, idem pour un booléen

Déclaration et affecation d'une variable

Déclaration

```
type nom_de_variable ;
```

Affectation

```
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 reel
```

C++

```cpp
a = 2; // ERREUR: a n'existe pas
int a; // declaration
a = 2; // affectation
int a = 2; // d
float b = 3.4;
double e, f;
bool c = true, d = false;
```

Les types natifs

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	$\mathbb{R}$	3.4E +/- 38	
double	$\mathbb{R}$	1.7E +/- 308	
bool	$\{0, 1\}$	false (0)	true (1)

Exemple

C++

```cpp
int i; // Definition
i = 2; // Affectation
cout << i << " " ; // Affiche "2"

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

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

int n = 5, o = n, p = INT_MAX ; // Initialisations
cout << n << " " << o << " " << p;

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

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

Conversions et règles de calcul

Opérateurs mathématiques

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

C++

```
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
```

Plan de la séance

Expressions booléennes

Une expression booléenne est une affirmation, elle est soit **VRAIE** soit **FAUSSE**.

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

C++

```
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 {...}`

C++

```
int annee;
cout << "Une annee ?" << endl;
cin >> annee;

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

C++

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

C++

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

```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
```

C++

```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
```

C++

```cpp
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 xrange(10):
    print(a)

# c'est equivalent a :

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

C++

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

// c'est equivalent a :

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

Sortir des boucles

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

C++

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

C++

```
/*
Sortir avant le terme d'un for
*/
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 :

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

Sens mathématique

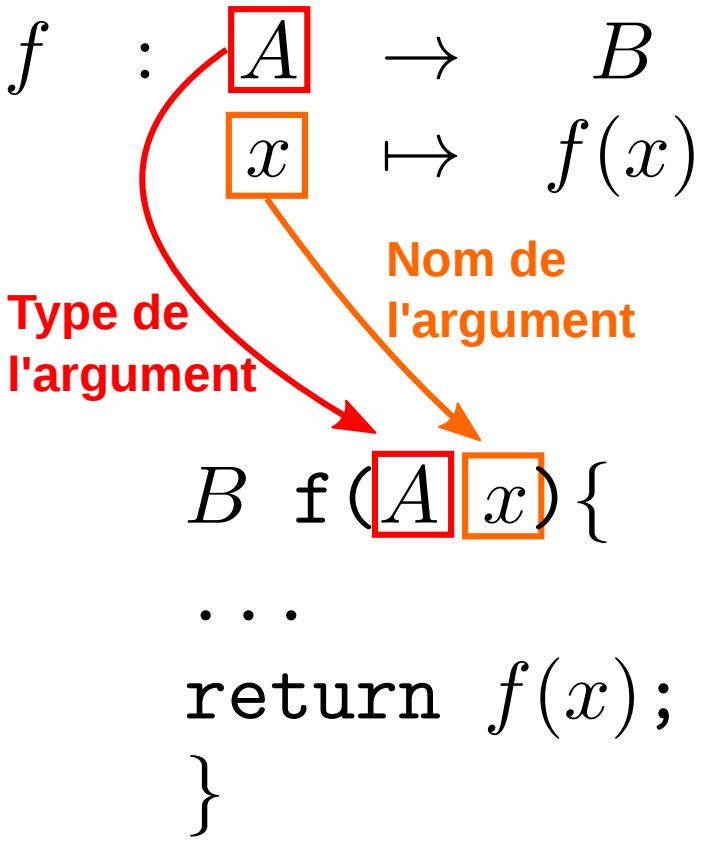
En C++, le concept de fonction est le même qu'en mathématiques :

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

$$\begin{array}{l} B \texttt{ f(} A \ x\texttt{)\{} \\ \texttt{...} \\ \texttt{return } f(x)\texttt{;} \\ \texttt{\}} \end{array}$$

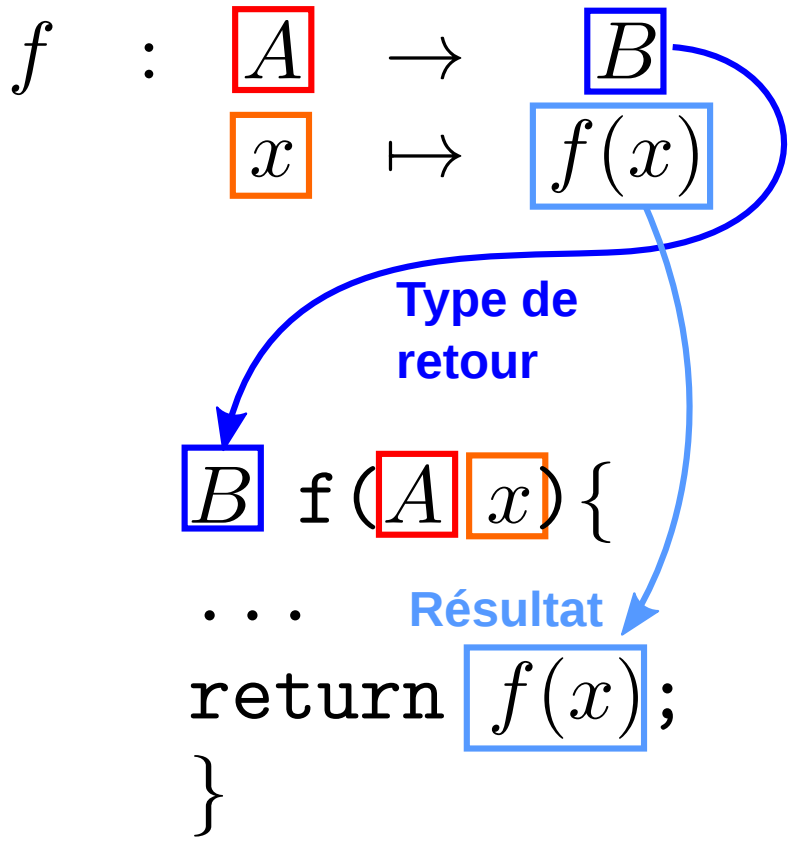
Sens mathématique

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



Sens mathématique

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



Exemple

La fonction signe

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

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 :

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

C++

```
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

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

C++

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

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

C++

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

C++

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

Le passage par référence

Le passage par référence est une solution aux problèmes précédents. Il autorise la modification de l'argument concerné.

Syntaxe

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

Passage par référence : exemples

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

C++

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

C++

```
int main(){
    int x, y;
    f(x, y);
    cout << x << " " << y << endl;
    // affiche 3 5
}
```

- Modifier les arguments

C++

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

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

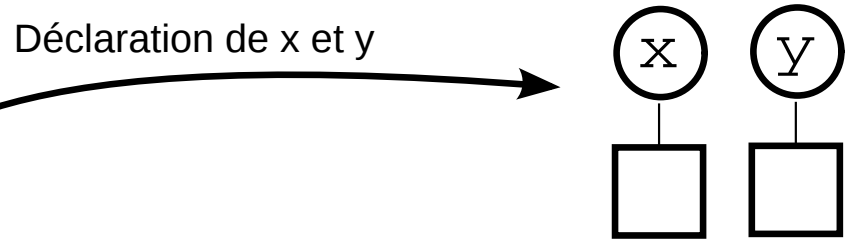
Fonctionnement de la fonction `switch_1`

```
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`

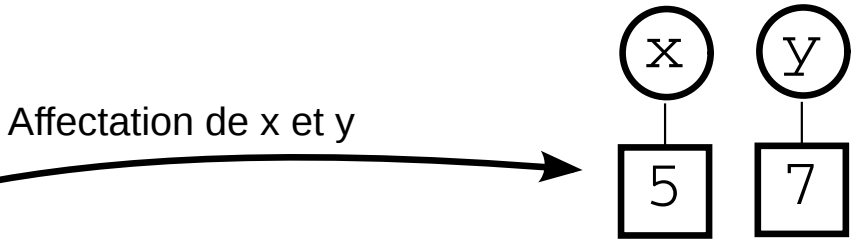
```
int main(){
  ...
  double x,y;
  x = 5;
  y = 7;
  switch_1(x,y);
}
```



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);
}
```

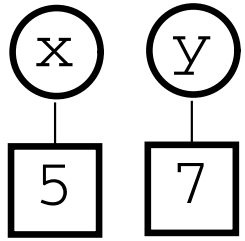


Passage par référence : zoom sur la mémoire 1

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;
}
```

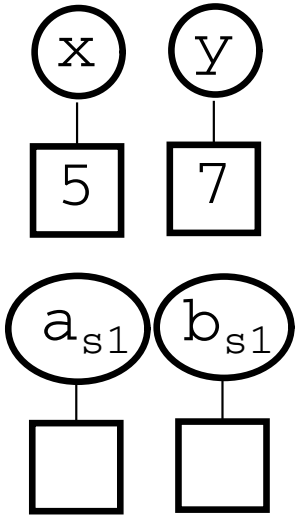


Passage par référence : zoom sur la mémoire 1

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;
}
```

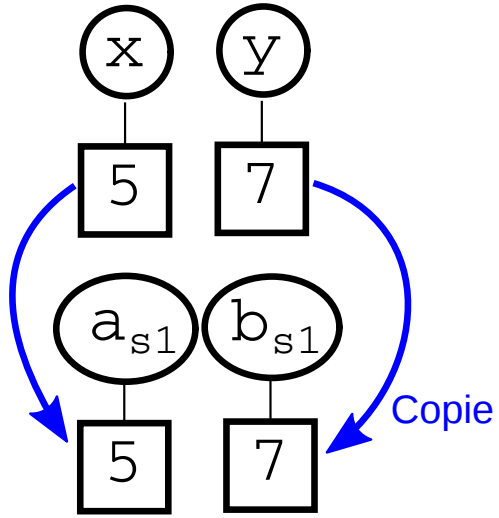


Passage par référence : zoom sur la mémoire 1

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;
}
```

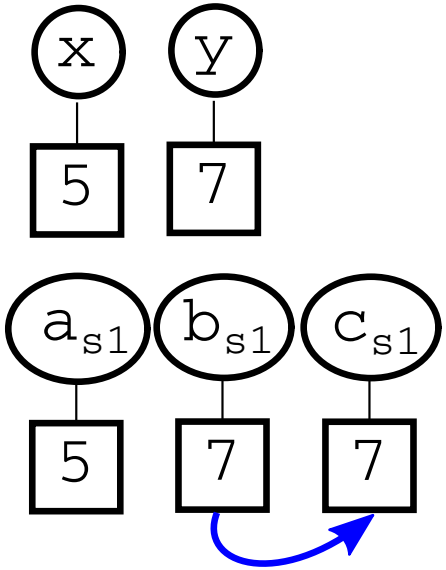


Passage par référence : zoom sur la mémoire 1

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;
}
```

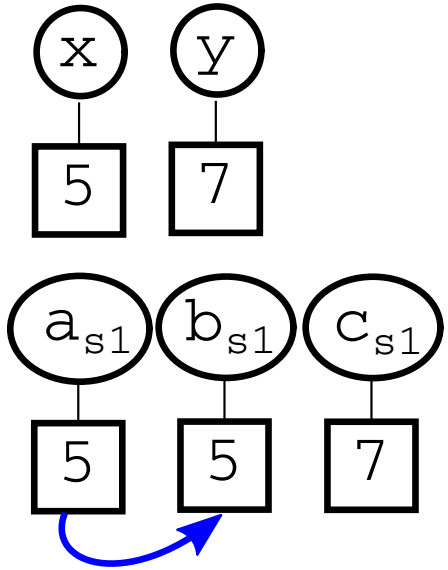


Passage par référence : zoom sur la mémoire 1

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;
}
```

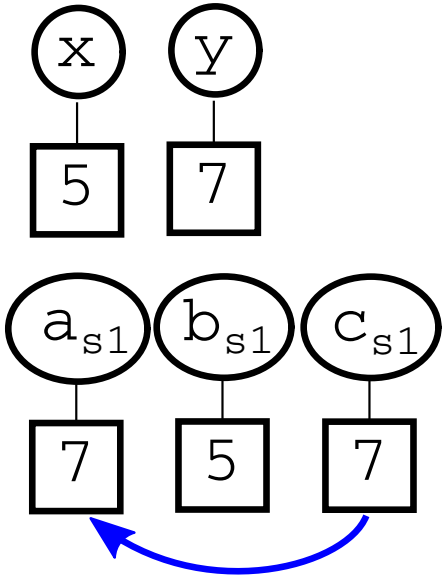


Passage par référence : zoom sur la mémoire 1

Fonctionnement de la fonction `switch_1`

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

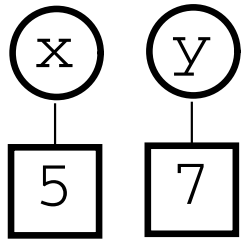
```
void switch_1(double a,
      double b){
   double c=b;
   b=a;
   a=c;
}
```



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

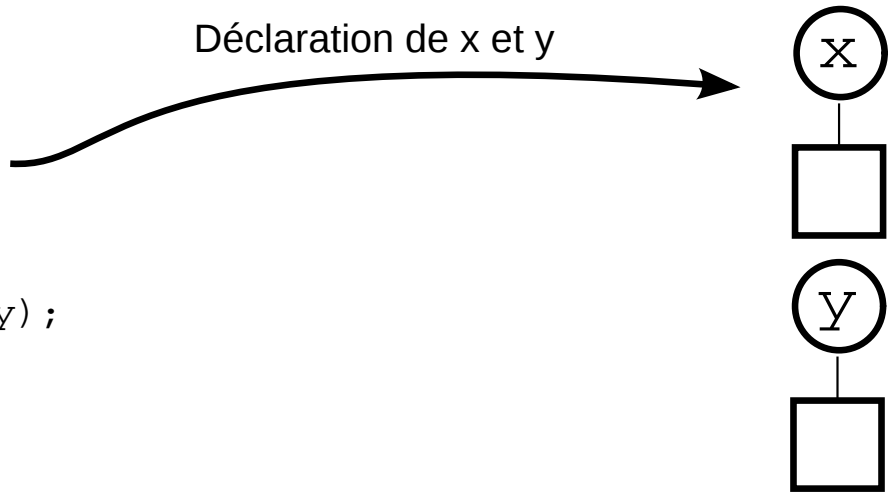
Fonctionnement de la fonction `switch_2`

```
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`

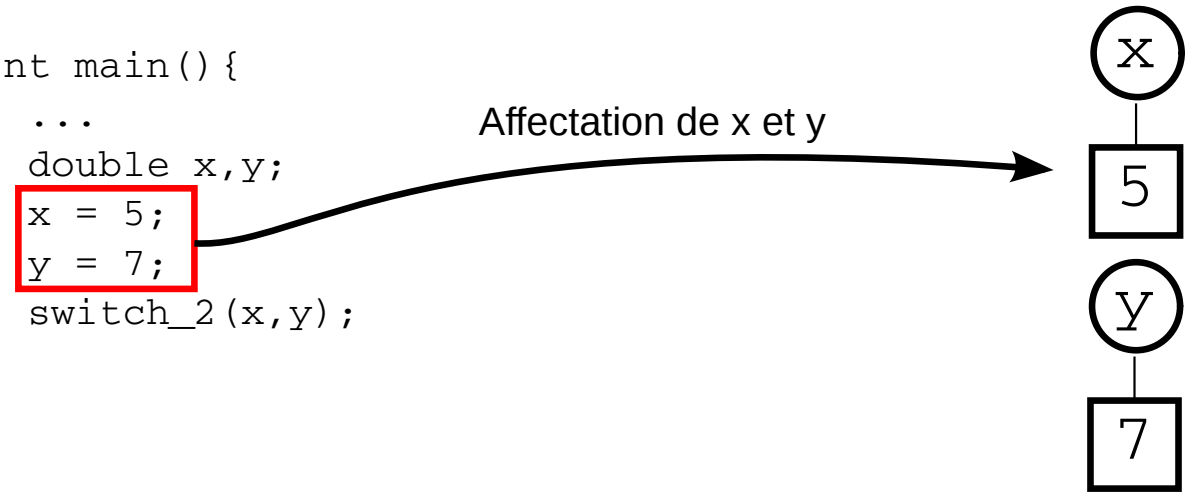
```
int main(){
  ...
  double x,y;
  x = 5;
  y = 7;
  switch_2(x,y);
}
```



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);
}
```

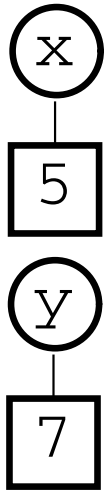


Passage par référence : zoom sur la mémoire 2

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;
}
```

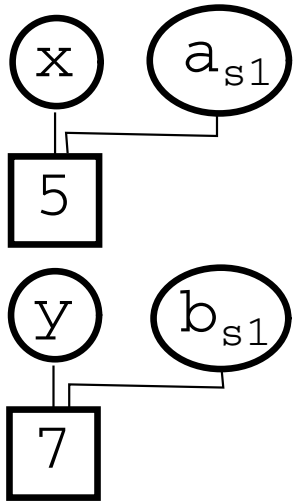


Passage par référence : zoom sur la mémoire 2

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;
}
```



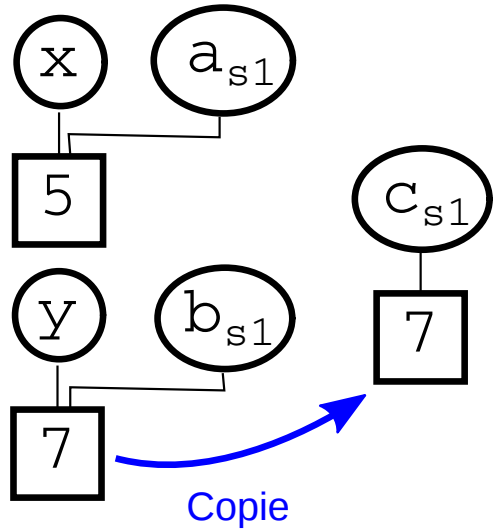
Partage de la mémoire

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;
}
```

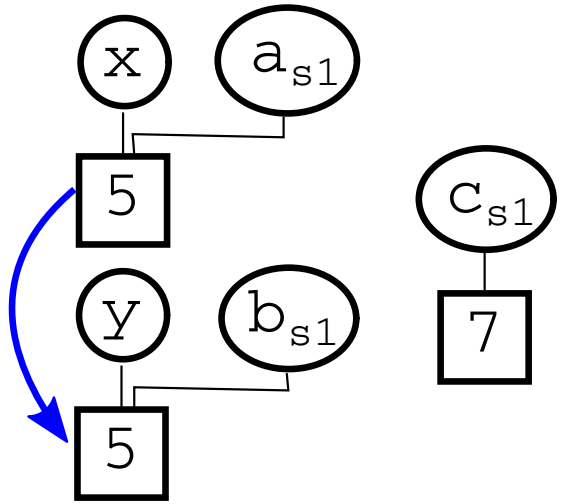


Passage par référence : zoom sur la mémoire 2

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;
}
```

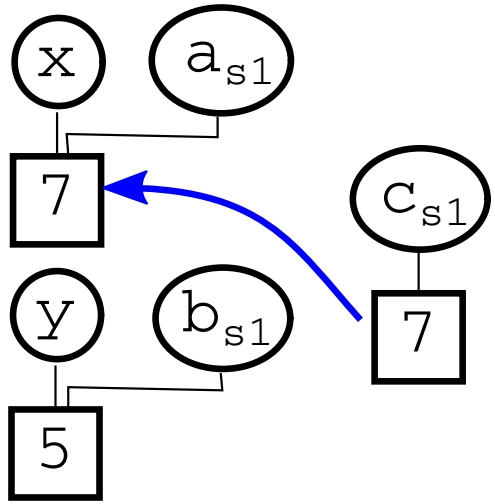


Passage par référence : zoom sur la mémoire 2

Fonctionnement de la fonction `switch_2`

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

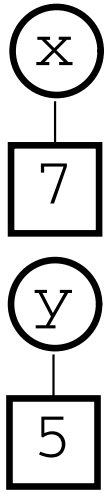
```
void switch_1(double & a,
      double & b){
   double c=b;
   b=a;
   a=c;
}
```



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);
}
```



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.

C++

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

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

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

C++

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

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

// OK
// Meme nom
// arguments differents
```

Portée, déclaration

Appel de fonction

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

C++

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

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

C++

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

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

Portée, déclaration

C++

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

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

C++

```
void g(int i); // declaration

void f(){
    g(3); //OK on a dit qu'une
}         // fonction g existe

//definition
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.

C++

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

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

C++

```
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.

C++

```
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)

TP de la semaine

Sujet

- Utilisation du débogueur
- Utiliser Imagine++ pour jouer à Pong !

Installation sur les portables

Instructions

Elles sont accessibles via le site du cours :

`http://imagine.enpc.fr/~monasse/Info/`

Pour travailler chez vous

Installer un IDE et Imagine++ sur vos portables.

En cas de problème

Séance d'installation lundi 26 septembre à 16h45

Prendre rendez-vous avec Pascal Monasse