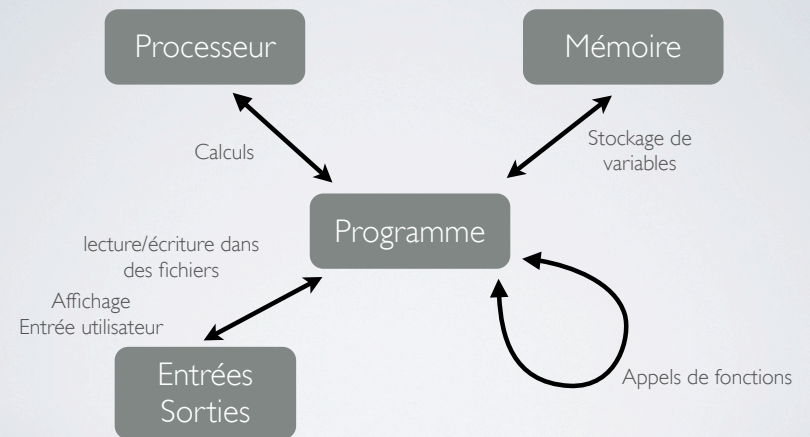


# COURS DE C++

## 2 - Instructions basiques

# TYPES D'INSTRUCTIONS



# INSTRUCTIONS PRÉPROCESSEUR

- Avant d'utiliser ces instructions il existe des instructions destinées au compilateur.
  - Ce sont les instructions pré-processeur.
- Elles commencent toujours par # et ne se terminent pas avec ;
- L'instruction standard est l'inclusion physique d'un autre fichier
    - `#include <nom_du_fichier>` quand il est dans un emplacement standard
    - `#include "nom_du_fichier"` quand il est dans un emplacement local

# EXPRESSIONS

- Les instructions manipulent des valeurs : nombres, caractères, objets, ...
- Valeurs immédiates : `3.04`, `2`, `«acbd»`, `'a'`
- Variables : `x`, `y`, `foo`, `bar`
- Fonctions : `f(3,2)`
- Expressions : `x+(3/2)`, `y == 2`, ...

# TYPES

- Les expressions sont construites en utilisant des opérateurs définis sur des types
- Trois classes de types :
  - Les types entiers : bool, char, short, int, ...
  - Les types flottants : float, double, ...
  - Les types composés : tableaux, struct, objets, ...

# TYPES ENTIERS

- représentent un sous-ensemble contigu des relatifs
- `bool` : 0, 1
- `char` : -128 à 127 `unsigned char` : 0 à 255
- `short` : -32768 à 32767 et `unsigned short` : 0 à 65535
- `int` : -2147483648 à 2147483647 et `unsigned int` : 0 à 4294967296
- d'autres, par exemple : entiers sur 64bits, caractères unicode, ...

# OPÉRATEURS SUR TYPES ENTIERS

- Opérateurs arithmétiques principaux : + - \* / %
  - attention la division est entière :  $x = (x/y)*y + (x \% y)$
  - Précédence : % > \* / > + - 5%3\*2-1 = ((5%3)\*2)-1
- Opérateurs de comparaison : == != < > <= >=
- Opérateurs logiques : || or && and ! not

# TYPES FLOTTANTS

- représentent de manière approchée un segment de rationnels
- ce sont des couples d'entiers (m, e), mantisse et exposant
  - (5421, 12) =  $5.421 \times 10^{12}$
- `float` : sur 32 bits (comme les int), m sur 24, e sur 8
- `double` : sur 64 bits, m sur 53, e sur 11
- `long double` : sur 80 bits, m sur 65, e sur 14

## OPÉRATIONS SUR TYPES FLOTTANTS

- Opérations de calcul : + - \* /
  - Attention ce n'est pas le même / :  $2.0/3.0 = 0.666667$
- Beaucoup de fonctions standards définis dans math.h (cmath) :
  - `#include <math.h>` ou `#include <cmath>`
  - p.ex.: `sqrt`, `cos`, `sin`, ...
  - en général définies pour des doubles, mais
    - `sinf` : pour float, `sinl` pour long double (idem pour les autres)

## STABILITÉ DES OPÉRATIONS

- Il existe trois valeurs particulières de flottants : inf, -inf, nan
- $1.0/0.0 = \text{inf}$ ,  $-1.0/0.0 = -\text{inf}$ ,  $0.0/0.0 = \text{nan}$
- Elles se propagent,  $1+\text{inf} = \text{inf}$ ,  $x+\text{nan} = \text{nan}$ , ...
- Problème : toute suite de flottants qui tend vers 0.0 est stationnaire ! Donc ces valeurs peuvent apparaître vite.
- Moralité : Dès que l'on fait des calculs numériques il faut faire très attention. La précision ne fait que retarder le problème.
- Parole d'expert : ce sont des bugs horribles à tracer

## DÉLIMITEUR ET BLOCS

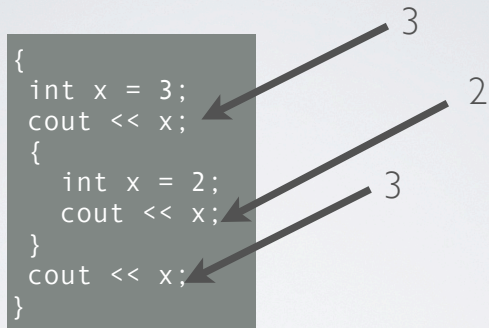
- Les instructions en C++ se terminent TOUJOURS par un ;
- On peut regrouper un ensemble d'instructions dans un bloc :
  - `{ instruction1; instruction2; ... }`
- Ne créez pas des blocs pour rien, c'est peu lisible.

## DÉCLARATION DE VARIABLES

- `type nom_de_la_variable;`
- `type nom_de_la_variable = valeur_initiale;`
- Les noms de variables sont de la forme `[a-zA-Z][a-zA-Z0-9_]*`
- Ils ne peuvent pas être des mots du langage : int, for, ...
- Deux styles en entreprise : `plusieurs_mots` ou `plusieursMots`

## PORTÉE

```
{
  int x = 3;
  cout << x;
  {
    int x = 2;
    cout << x;
  }
  cout << x;
}
```



Par défaut, une variable est accessible depuis sa déclaration jusqu'à la fin du bloc courant avec masquage potentiel des variables de même nom

## AFFECTATION DE VARIABLES

- `x = 2;`
  - Renvoie la valeur 2, on peut donc chaîner : `x = y = 2;`
- Plusieurs instructions abrégés :
  - `x++;` signifie `x = x+1;` et renvoie `x`
  - `++x;` signifie `x = x+1;` et renvoie `x+1`
  - `x *= 2;` signifie `x = x*2;` de même pour tous les opérateurs

## SORTIE SUR LA CONSOLE

- On utilise un flux de sortie (outstream).
- Les flux standards sont définis dans
  - `#include <iostream>`
- Le flux de sortie standard est `cout`
- Pour envoyer une valeur sur un flux `cout << valeur;`
- on peut chaîner les sorties : `cout << x; cout << y;` est équivalent à `cout << x << y;`
- `endl` est une valeur spéciale fin de ligne.
- les valeurs chaînées sont évaluées de la fin vers le début !

## ENTRÉE SUR LA CONSOLE

- On utilise un flux d'entrée (instream)
- le flux d'entrée standard est `cin`
- Pour lire une valeur on effectue `cin >> x;` qui va prendre la valeur entrée et la mettre dans la variable `x`
- On peut lire plusieurs valeurs d'affilée en chaînant