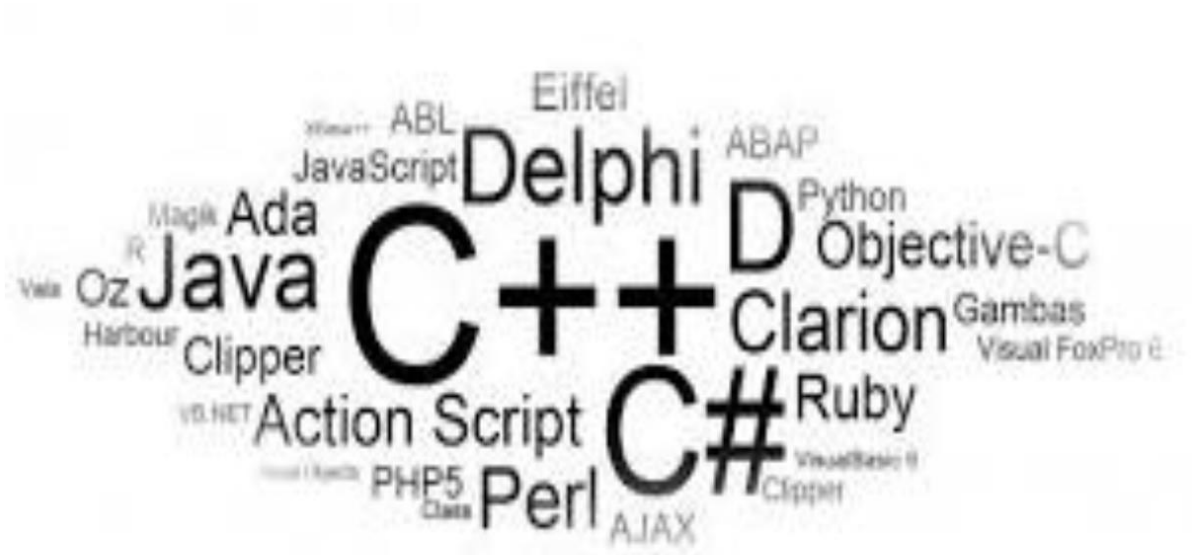
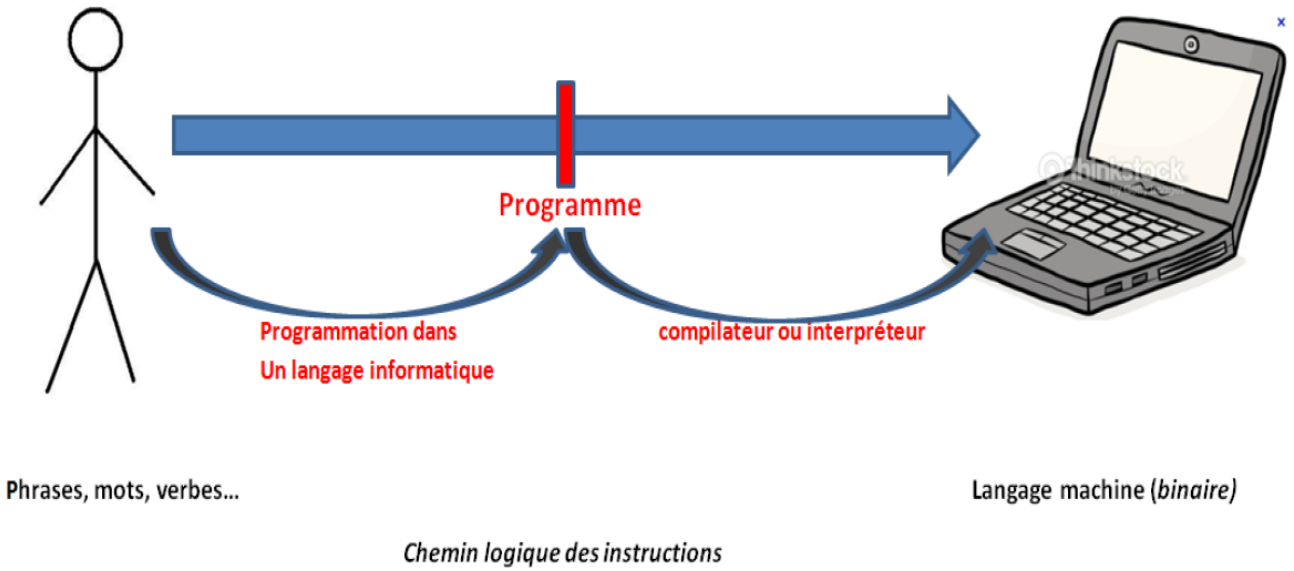
# LANGAGE C++

## INTRODUCTION




# Qu'est-ce qu'un langage de programmation

- Le **langage** est la capacité d'exprimer une pensée et de communiquer au moyen d'un système de signes doté d'une sémantique, et le plus souvent d'une syntaxe. Plus couramment, le langage est un moyen de communication.
- Un **langage de programmation** est un code de communication, permettant à un être humain de dicter des ordres (instructions) à une machine qu'elle devra exécuter.

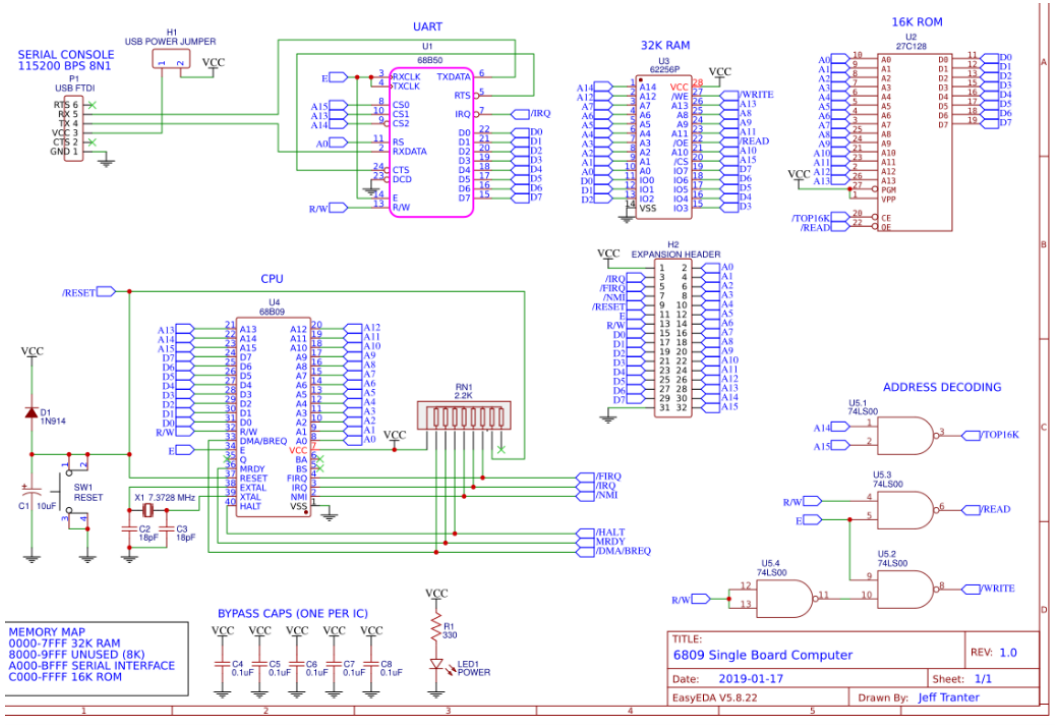



*Chemin logique des instructions*

# Généralités sur les ordinateurs

## Deux entités fondamentales

- **Microprocesseur** : Un microprocesseur est un circuit intégré complexe qui est capable d'interpréter et d'exécuter les instructions d'un programme. Schématiquement, on peut le considérer comme une puissance de calcul.
- **Mémoire** : permet de mémoriser des données.




L'ordinateur n'est pas une machine pensante et intelligente, capable de résoudre des problèmes.

- En fait, celui-ci n'est capable de rien si un programmeur ne lui fournit la liste des actions à exécuter.
- Les opérations élémentaires que peut exécuter un ordinateur sont en nombre restreint

# Généralités sur les ordinateurs

Les ordinateurs utilisent uniquement le 0 et le 1 :

- Tout ce que qu'on entend ou voit sur l'ordinateur – mots, images, nombres, films et même sons – est stocké à l'aide de ces deux chiffres uniquement.

- Tout programme exécuté par l'ordinateur est lui aussi codé par des 0 et des 1.

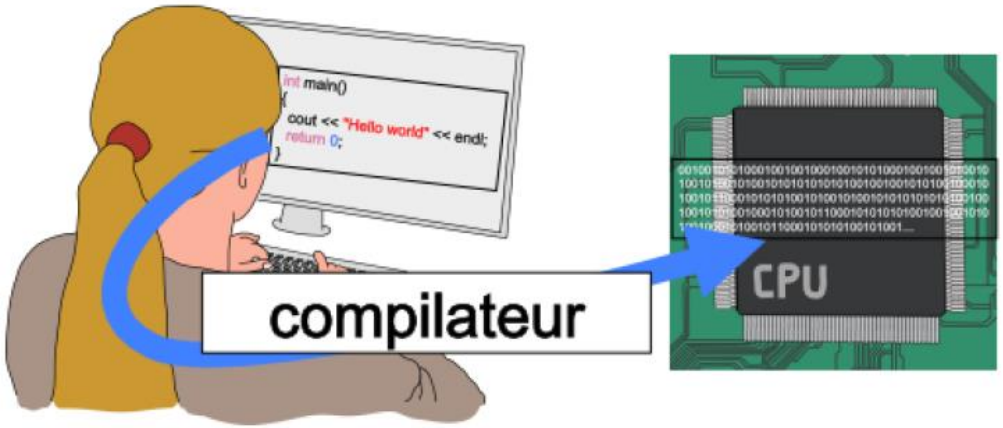
```
00011111011011101100010100000001
11011000111110110111011000101000
00001101100011111011011101100010
01100011111011011101100010100000
00111110110111011000101000000011
01011000011011000111110110111011
```

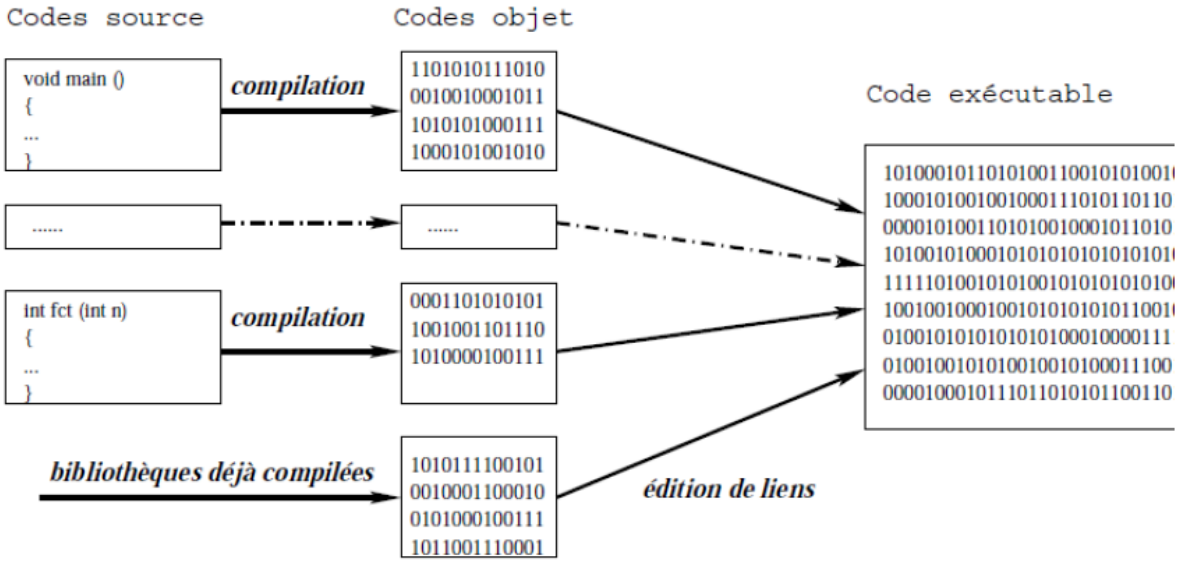
```
00:00000080  8D 07 F2 3A C9 66 9C 69 C9 66 9C 69 C9 66 9C 69
00:00000090  EE A0 E1 69 DA 66 9C 69 EE A0 F1 69 9B 66 9C 69
00:000000A0  4A 6E C1 69 CB 66 9C 69 0A 69 C1 69 C0 66 9C 69
00:000000B0  C9 66 9D 69 4F 66 9C 69 EE A0 F2 69 E4 66 9C 69
00:000000C0  EE A0 E6 69 C8 66 9C 69 EE A0 E0 69 C8 66 9C 69
00:000000D0  EE A0 E4 69 C8 66 9C 69 52 69 63 68 C9 66 9C 69
```

Les ordinateurs, quels qu'ils soient, ne sont fondamentalement capables de comprendre que quatre catégories d'ordres (en informatique d'**instructions**). Ces quatre familles d'instructions sont :

1. la lecture / écriture en mémoire
2. les tests
3. les boucles
4. les opérations mathématiques

# Le C++ est un langage compilé

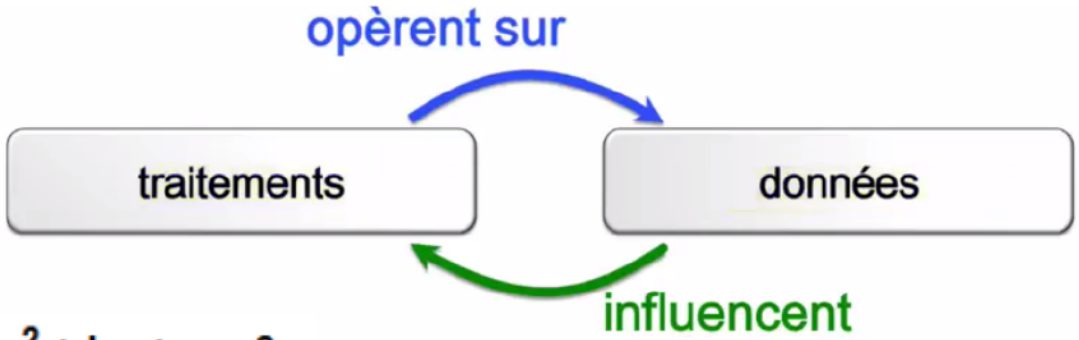


# Notion de programme

- Un programme est une suite d'instructions (d'opérations) compréhensible par l'ordinateur qui décrit ce qu'il devra exécuter.

- Une instruction dicte à l'ordinateur l'action nécessaire qu'il doit effectuer avant de passer à l'instruction suivante.



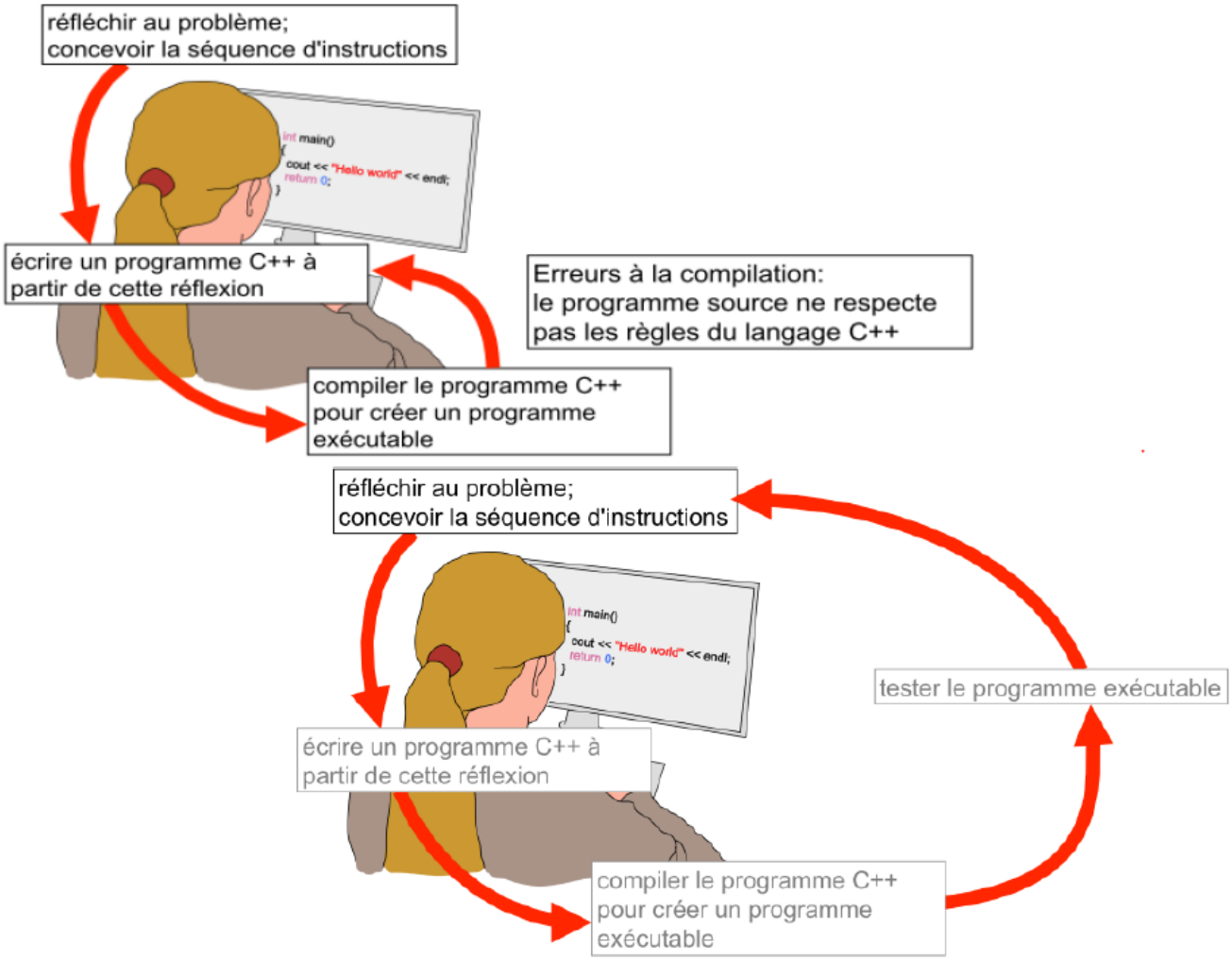

**Exemple :** $ax^2 + bx + c = 0$

1. Demander les coefficients a, b et c à l'utilisateur
2. Calculer delta
3. Si delta < 0 => **« pas de solution »**
4. Sinon si delta = 0 => **x=-b/a**
5. Sinon si deltat > 0 => **x' =( -b + √Δ ) / 2a et x' =( -b -√Δ ) / 2a**

**Remarque :**

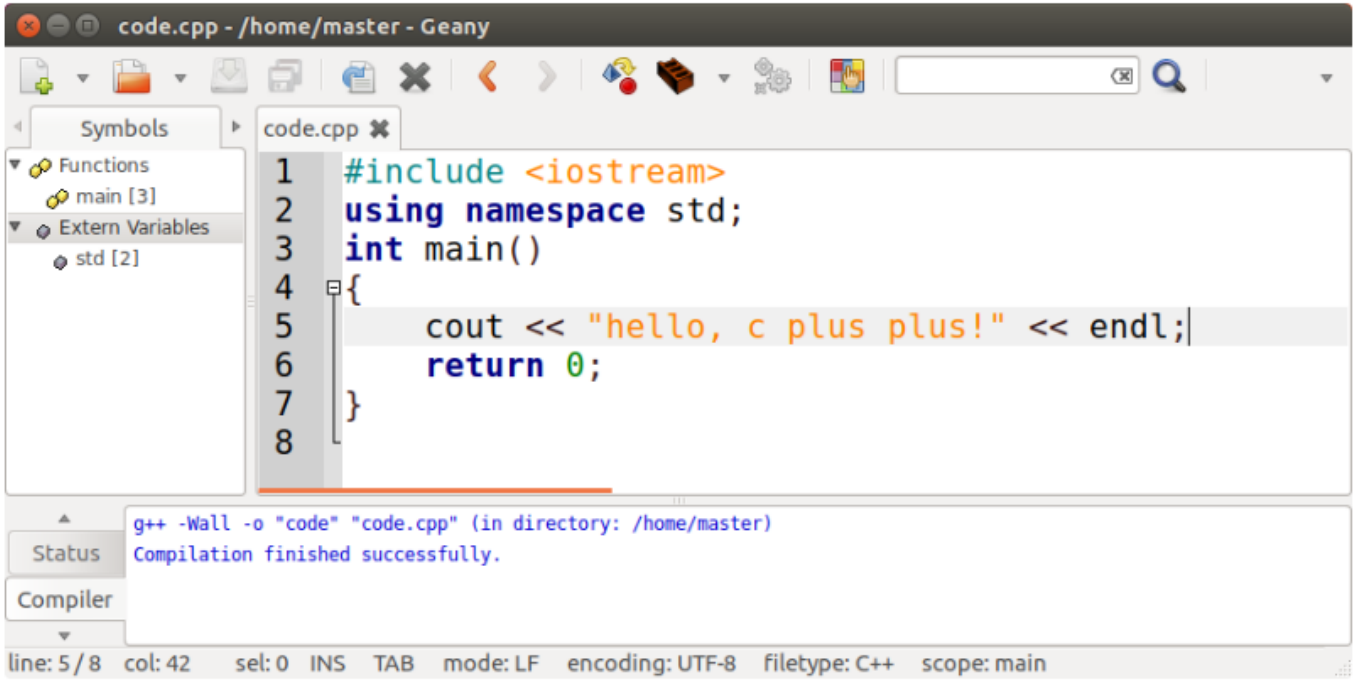
- expression : évaluer une expression arithmétique pour calculer delta
- variable : delta mémorise une valeur dont j'ai besoin à **plusieurs endroits**

# Etapes de réalisation d'un programme



# Premier programme

**Démo** : créer un projet

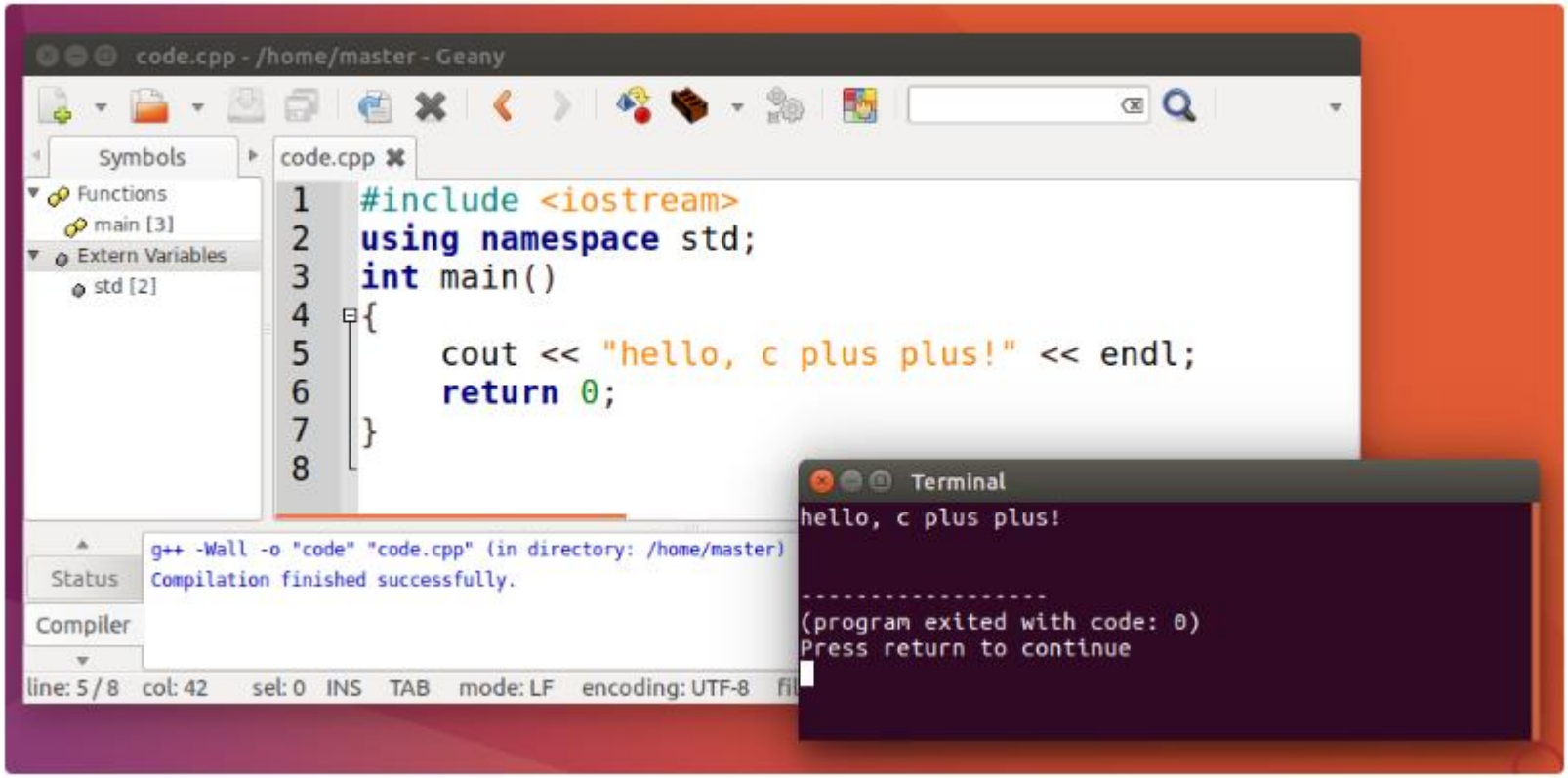



Les deux premières lignes seront étudiées plus tard

- int main() : Tous les programmes possèdent une fonction dénommée « main » ce qui signifie « principale », elle représente le point d'entrée.
- cout << "Hello world!" << endl ; pour l'instant nous considérons cette insruction comme la fonction d'affichage sur le terminal.

# Premier programme (suite)

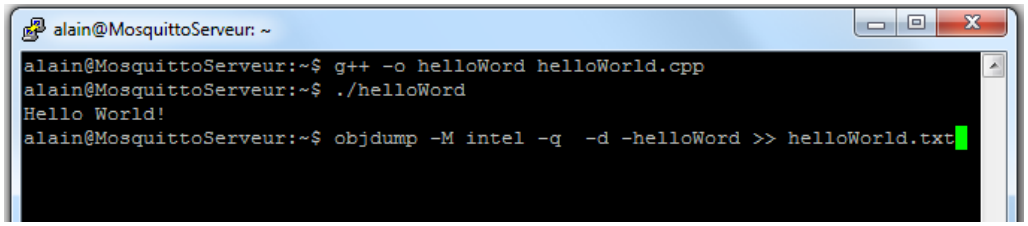
## Exécution du programme

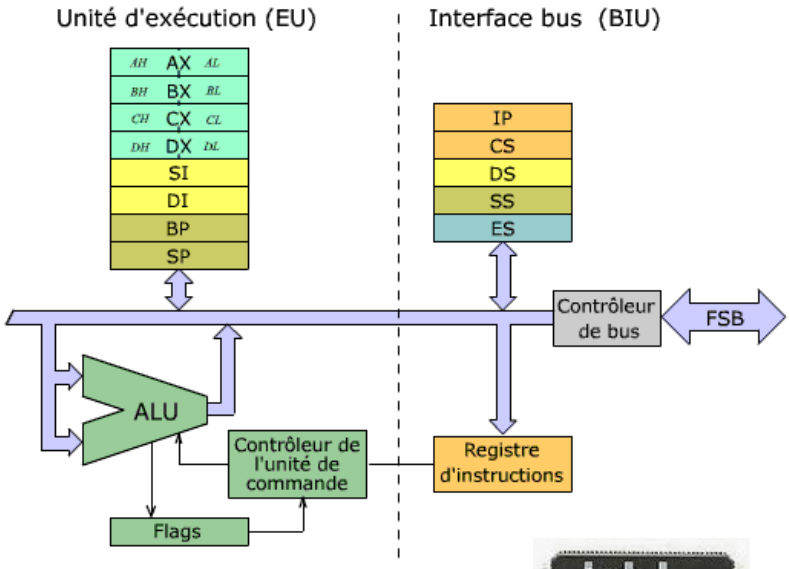


# C/C++ , assembleur et langage machine

```cpp
#include <iostream>

int main() {
    std::cout << "Hello World!" << std::endl ;
    return 0;
}
```





```
08048614 <main>:
 8048614:   55                      push   ebp
 8048615:   89 e5                   mov    ebp,esp
 8048617:   83 e4 f0                and    esp,0xfffffff0
 804861a:   83 ec 10                sub    esp,0x10
 804861d:   c7 44 24 04 80 87 04    mov    DWORD PTR [esp+0x4],0x8048780
 8048624:   08
 8048625:   c7 04 24 40 a0 04 08    mov    DWORD PTR [esp],0x804a040
 804862c:   e8 ff fe ff ff          call   8048530 <_ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc@plt>
 8048631:   c7 44 24 04 50 85 04    mov    DWORD PTR [esp+0x4],0x8048550
 8048638:   08
 8048639:   89 04 24                mov    DWORD PTR [esp],eax
 804863c:   e8 ff fe ff ff          call   8048540 <_ZNSolsEPFRSoS_E@plt>
 8048641:   b8 00 00 00 00          mov    eax,0x0
 8048646:   c9                      leave
 8048647:   c3                      ret
```

# Notion de variable

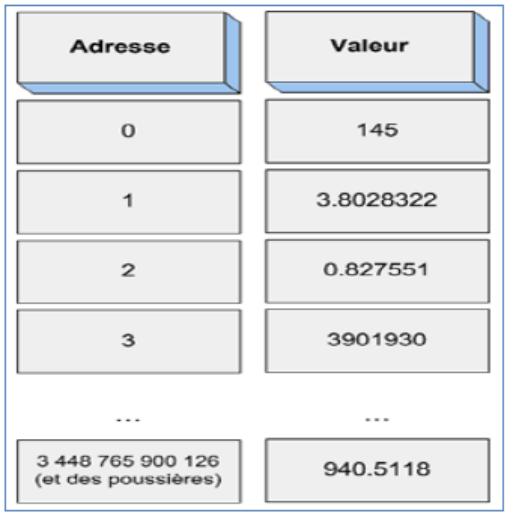
**Une variable, en programmation, est un espace mémoire**

**Un ordinateur à plusieurs types de mémoire :**

1. Les registres : une mémoire ultrarapide située directement dans le processeur – mémoire temporaire
2. La mémoire cache : elle fait le lien entre les registres et la mémoire vive – mémoire temporaire
3. **La mémoire vive** : c'est la mémoire avec laquelle nous allons travailler le plus souvent – mémoire temporaire
4. **Le disque dur** : C'est là qu'on enregistre les fichiers – mémoire persistante

**Une variable est stockée en mémoire vive (RAM).**

**Schéma de la mémoire vive**




La mémoire est un ensemble de "positions binaires" nommées bits. Les bits sont regroupés en octets (8bits), et chaque octet est repéré par son adresse.
Deux colonnes :
- Adresses : identifie une zone mémoire
- A chaque adresse, on peut stocker une valeur (en binaire)

Toute information, quelle que soit sa nature, devra être codée sous forme binaire.
Il ne suffit pas de connaître le contenu d'un emplacement de la mémoire pour être en mesure de lui attribuer une signification mais de connaitre aussi son type.

# Définition d'une variable

**Une variable est un triplé composé**

1. d'un identificateur : nom par lequel la donnée est désignée
2. d'un type : définit de quel « genre » est la donnée contenue dans la variable
3. le contenu de la variable : sa valeur

## Nommer une variable, règles

- les noms de variables sont constitués de lettres, de chiffres et du tiret-bas _ uniquement.
- Votre nom de variable doit commencer par une lettre.
- Vous n'avez pas le droit d'utiliser des accents
- Les espaces sont interdits.

## Les types de base

| Type | Signification |
|---|---|
| bool | Booléen : vrai (true) ou faux (false) |
| char | Caractère |
| int | Entier |
| float ou double | Réel |
| String | Chaine de caractères (mot ou phrase) |

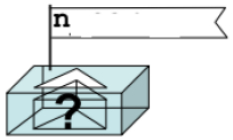
**Remarque :** La taille en octet des types de variable dépend du microprocesseur sur lequel nous développons (8bits, 16bits, 32 bits, etc…).

# Déclaration de variable

Permet à l'ordinateur de **réservé de l'espace mémoire** (dont l'espace dépend du type) et donc de **créer la variable**.

**Déclaration sans initialisation**

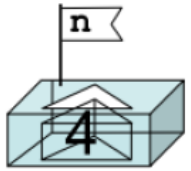
```
int n;
String s ;
double d ;
```




**Note** : A la déclaration de la variable, le microprocesseur réserve un espace mémoire pour la variable. Votre variable prend la valeur qui se trouvait là avant dans la mémoire, cette valeur peut être n'importe quoi !

**Déclaration avec initialisation**

```
int n(4);                ou    int n = 4 ;
String s("Bonjour") ;    ou    String s = "Bonjour" ;
double d(4.0) ;          ou    d = 4.0 ;
```




**Règle : Il faut toujours initialiser une variable avant d'utiliser la valeur qu'elle contient !**

```
int main()
{
    int n;
    cout << "Valeur variable = " << n << endl;
    return 0;
}
```

Code::Blocks | Search results | Cccc | Build log | **Build messages** | CppCheck

| File | Line | Message |
|---|---|---|
| | | === Build: Debug in premier (compiler: GNU GCC Compiler) === |
| C:\Tophe\Langa... | | In function 'int main()': |
| *C:\Tophe\Langa... 8* | | *warning: 'n' is used uninitialized in this function [-Wuninitialized]* |
| | | === Build finished: 0 error(s), 1 warning(s) (0 minute(s), 0 second(s)) === |

```
Valeur variable n = 4285822

Process returned 0 (0x0)   execution time : 0.020 s
Press any key to continue.
```

# Opérateur d'affectation

**Schéma d'une affectation**: *nom_de_variable* = ***expression;***

N'oubliez pas le point-virgule ; à la fin

**Exemple** :

```
int n(4) ;
int n_carre ;
n_carre = n * n;          // est une affectation.
```

**Une affectation est une instruction qui permet de modifier la valeur d'une variable.**
**=> Attention, une affectation n'est pas une égalité mathématique**
**Règle générale : une expression calcule une valeur, qui doit être de même type que la variable.**

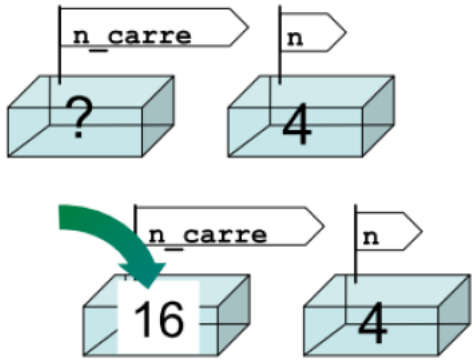
**L'exécution d'une affectation se décompose en deux temps** :

**1. L'expression à droite du signe = est évaluée**

- n * n est évaluée avec la valeur de n au moment de l'exécution.
- L'étoile * représente la multiplication, n * n vaut donc 4 × 4 = 16

**2. La valeur de l'expression est stockée dans la variable à gauche du signe =**

- L'ancienne valeur de n_carre est perdue




Autres exemples d'expressions :
- 4
- n * n
- n * (n + 1) + 3 * n – 2

# Une affectation n'est pas une égalité mathématique

```
a = b;
b = a;
```

copie la valeur de b dans a

copie la valeur de a dans b

```
a = 5;
b = a + 1;
a = 2;
```

donne à b la valeur de a+1, c'est-à-dire 6.

donne à a la valeur 2, sans que b ne soit modifiée!
b contient donc toujours 6.

a = a + 1;
=> signifie « calculer l'expression de a + 1 et ranger le résultat dans a. Cela revient à augmenter de 1 la valeur de a »

## Déclaration de constantes - le mot-clé *const*

```
const int nConstante(8) ;      ou      const int nConstante = 8 ;
```

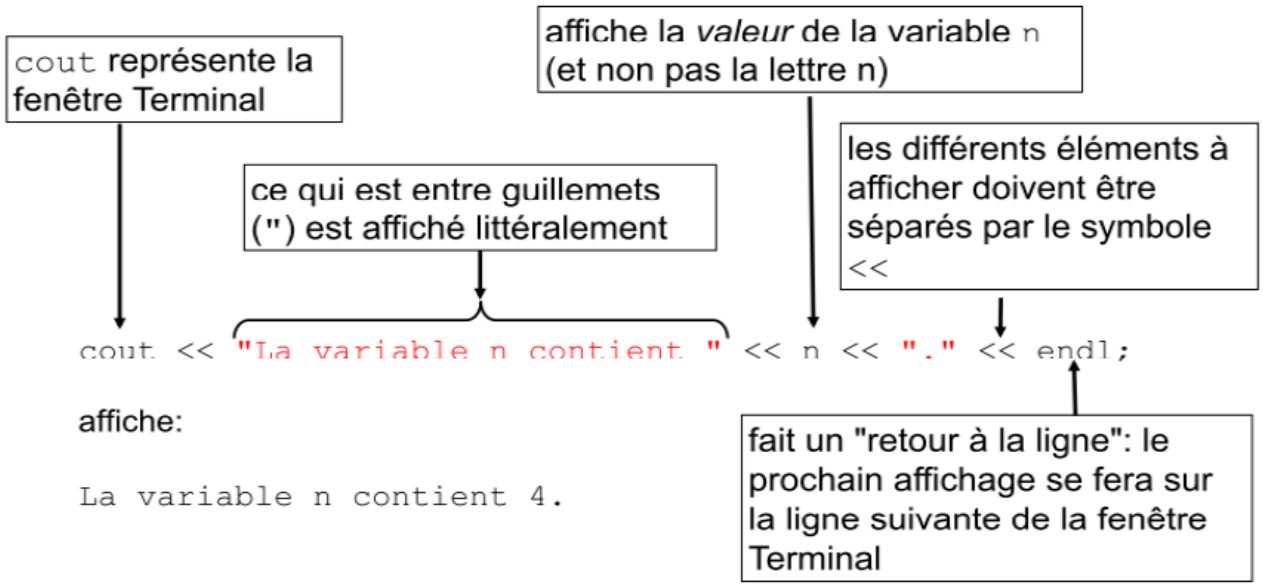
```
const double vitesse_de_la_lumiere(299792.458);
vitesse_de_la_lumiere = 100; // erreur  de compilation!
```

Dans ce cas, on ne peut plus modifier la variable:

```
int main()
{
    const int nConstante = 8 ;
    nConstante = 10 ;
    return 0;
}
```

| File | Line | Message |
|---|---|---|
| | | === Build: Debug in test (compiler: GNU GCC Compiler) === |
| C:\Top... | | In function 'int main()': |
| C:\Top... | 8 | error: assignment of read-only variable 'nConstante' |
| | | === Build failed: 1 error(s), 0 warning(s) (0 minute(s), 0 second(s)) === |

# Ecrire la valeur d'une variable à l'écran




Le flot **cout** est un flot de sortie prédéfini, connecté à la sortie standard stdout (l'écran)

**On peut aussi afficher une expression :**

```
cout << "Le double de " << n << " est " << 2 * n << "." << endl;
```

expression

**Remarque** :
cout et endl sont des mots réservé de la bibliothèque standard std (using namespace std), Nous pouvons les nommer std::cout et std::endl

```
#include <iostream>
int main(){
    std::cout << "Message sans namespace défini" << std::endl;
    return 0;
}
```

**Démo** : Déroulement du programme pas-à-pas

```
int n(4);
int n_carre;
n_carre = n * n;
cout << "La variable n contient " << n << "." << endl;
cout << "Le carre de " << n << " est " << n_carre << "." << endl;
cout << "Le double de n est " << 2 * n << "." << endl;
```

**Exercices** : Qu'affichent ces programmes ?

```
#include <iostream>
using namespace std;
int main(){
    int a(2);
    int b(1);
    b = a * (b + 2);
    cout << a << ", " << b << endl;
}
```

A: a, b
B: 1, 2
C: 2, 1
D: 2, 6

```
#include <iostream>
using namespace std;
int main(){
    int a(5);
    int b(a + 3);
    a = 1;
    cout << a << ", " << b << endl;
}
```

A: 5, 4
B: 1, 1
C: 1, 4
D: 1, 8

```cpp
#include <iostream>
using namespace std;
int main(){
    int a(1);
    int b(2);
    a = b;
    b = a;
    cout << a << ", " << b << endl;
}
```

A: 1, 1
B: 1, 2
C: 2, 2
D: 2, 1

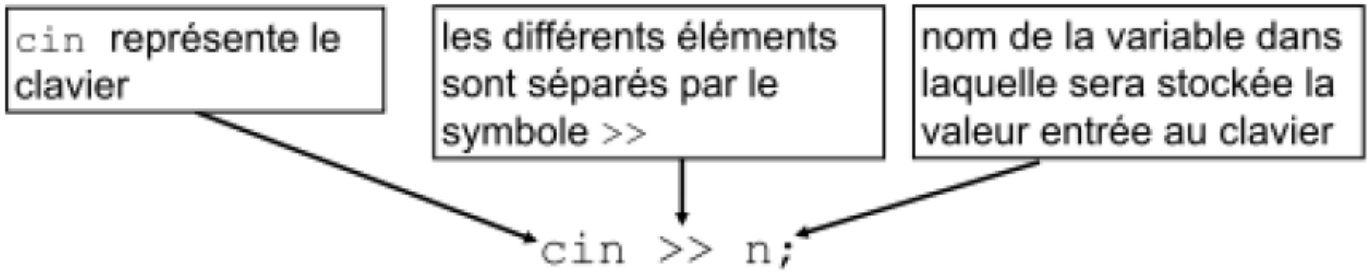
**Exercice**: Comment échanger la valeur de deux variables a et b ?

```
#include <iostream>
using namespace std;
int main(){
    int a(1);
    int b(2);
    a = b;
    b = a;
    cout << a << ", " << b << endl;
}
```

A: 1, 1
B: 1, 2
C: 2, 2
D: 2, 1

**Exercice**: Comment échanger la valeur de deux variables a et b ?

# Lire à partir du clavier




- **Attention**, uniquement des noms de variables peuvent figurer à droite du symbole >>
- Le flot **cin** est un flot d'entrée prédéfini, connecté à l'entrée standard stdin (le clavier).

**Démo** : exécuter pas à pas la séquence de code suivante (fonction bloquante)

```
int main(){
      int n;
      cout << "Saisr une valeur entiere pour n:";
      cin >> n;
      cout << "La variable n contient " << n << "." << endl;
}
```

**Lire plusieurs valeurs à la suite**

```
int main(){
    int n1 = 0, n2 = 0, n3 = 0;
    cout << "Saisir deux entiers" << endl;
    cin >> n1 >> n2 >> n3;
    cout << n1 << ", " << n2 << ", " << n3 << endl;
}
```

```
Saisir deux entiers
1
2
3
1, 2, 3

Process returned 0 (0x0)   execution time : 5.174 s
Press any key to continue.
```

# Expressions et opérateurs

A droite du signe égal dans une affectation se trouve une **expression**:

```
nom_de_variable = expression;
```

## Une expression

- calcule une valeur, qui doit être de même type que la variable.
- une expression possède son propre type
- peut-être simplement une valeur littérale (ex : n=4 ou d=3.14)
- ou une formule qui met en oeuvre des opérateurs:

  n * n

  n * (n + 1) + 3 * n - 2

Les valeurs littérales et leurs propres types :

- 1 est de type int
- 1.0 est de type double

Note : notation scientifique : 2e3 pour 2×10 3 double d(2e3) ;

## Les opérateurs

- sont des symboles qui permettent de manipuler des variables, c'est-à-dire effectuer des opérations, les évaluer.

Les principales familles d'opérateurs:

- Les opérateurs arithmétiques
- Les opérateurs de test
- Les opérateurs logiques booléens

# Les opérateurs arithmétiques

**=** affectation
**+** addition
**-** soustraction
***** multiplication
**/** division
**%** reste de la division (modulo, ne s'applique qu'à des opérandes de type entier).

**Note : division**

si la division se fait entre **deux** entier (int), il s'agit de la division entière :

1 / 2 **vaut** 0 (1 = **0** x 2 + 1) 5 / 2 **vaut** 2 ( 2 = **2** x 2 + 1)
1 / 2.0 **vaut** 0.5 (1 convertit en double 1.0)

**Exemple :**

```
int main(){
      double x;
      x = 1/2;
      // l'expression 1/2 est évaluée, elle vaut 0,
      // la valeur 0.0 est affectée à x
      cout << "x = " << x << endl;
}
```

```
x = 0

Process returned 0 (0x0)   execution time : 0.010 s
Press any key to continue.
```

**Exercice** : Quelle est la valeur de la variable moyenne ?

```
int main(){
    int note1(4), note2(5);
    double moyenne((note1 + note2) / 2);
    cout << "moyenne = " << moyenne << endl;
}
```

**Solutions** :

```
int main(){
    int note1(4), note2(5);
    double moyenne(note1 + note2);
    moyenne /= 2;
    cout << moyenne << endl;
}
```

ou

```
int main(){
    int note1(4), note2(5);
    double moyenne((note1 + note2)/2.0);
    cout << moyenne << endl;
}
```

**Opérateurs raccourcis** : += , -= , *= , /=

Exemple : a = a + b ; est équivalent à a+=b ;

**Pour les variables entières**
il existe un opérateur d'incrémentation (++n) et un autre de décrémentation (--m) :

++n ; est équivalent à n = n + 1 ;
--m ; est équivalent à m = m - 1 ;

# Affectation d'une valeur décimale à une variable entière :

```
int main(){
    double x(1.5);
    int n;
    n = 3 * x;
    cout << "n = " << n << endl;
}
```

```
n = 4

Process returned 0 (0x0)   execution time : 0.005 s
Press any key to continue.
```

**Règle générale :** Le langage C++ est dit « fortement typé », il demande que la variable et l'expression d'une opération d'affectation soit de même type (tant que faire se peut).

# Les fonctions mathématiques

La bibliothèque standard du C++ cmath fournit les fonctions mathématiques usuelles.

**Exemple**

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

int main()
{
  double angle;
  double s;
  angle = 10 * 3.14159 / 180;
  s = sin(angle);
  cout << "Le sinus d'un angle de 10 degrés est " << s << "." << endl;
}
```

Ajouter cette ligne pour inclure la bibliothèque mathématique et ainsi utiliser les fonctions mathématiques.

$$\text{angle en radians} = \frac{\pi \text{ angle en degrés}}{180}$$

```
Le sinus d'un angle de 10 degrés est 0.173648.

Process returned 0 (0x0)   execution time : 0.010 s
Press any key to continue.
```

# Quelques fonctions

- `sin` les fonctions trigonométriques fonctionnent en radians
- `cos`
- `tan`
- `asin` sinus inverse ou arc sinus
- `acos`
- `atan`
- `atan2` `atan2(y, x)` fournit la valeur de l'arc-tangente de `y / x`
- `sinh` sinus hyperbolique ou *sh*
- `cosh`
- `tanh`
- `exp`
- `log` logarithme népérien ou *ln*
- `log10` logarithme à base 10 ou *log*
- `pow` `pow(x, y)` fournit la valeur de $x^y$
- `sqrt` racine carrée
- `ceil` `ceil(x)` renvoie le plus petit entier qui ne soit pas inférieur à `x`: `ceil(2.6) = 3`
- `floor` `floor(x)` renvoie le plus grand entier qui ne soit pas supérieur à `x`: `floor(2.6) = 2`
- `abs` valeur absolue

# Annexe Mots-clés

C++ possède, par rapport à C, les mots-clés supplémentaires suivants :

| | | | |
|---|---|---|---|
| bool | catch | class | const_cast |
| delete | dynamic_cast | explicit | export |
| false | friend | inline | mutable |
| namespace | new | operator | private |
| protected | public | reinterpret_cast | static_cast |
| template | this | true | throw |
| try | typeid | typename | using |
| virtual | | | |

Voici la liste complète des mots-clés de C++. Ceux qui existent déjà en C sont en romain, ceux qui sont propres à C++ sont en italique. À simple titre indicatif, les mots-clés introduits tardivement par la norme ANSI sont en gras (et en italique).

| | | | | |
|---|---|---|---|---|
| asm | auto | ***bool*** | break | case |
| *catch* | char | *class* | const | ***const_cast*** |
| continue | default | *delete* | do | double |
| ***dynamic_cast*** | else | enum | *explicit* | ***export*** |
| extern | ***false*** | float | for | *friend* |
| goto | if | *inline* | int | long |
| ***mutable*** | ***namespace*** | *new* | *operator* | *private* |
| *protected* | *public* | register | ***reinterpret_cast*** | return |
| short | signed | sizeof | static | ***static_cast*** |
| struct | switch | *template* | *this* | *throw* |
| ***true*** | *try* | typedef | ***typeid*** | ***typename*** |
| union | unsigned | ***using*** | *virtual* | void |
| volatile | wchar_t | while | | |