Introduction à la programmation
Travaux pratiques: séance d'introduction
INFO0201-1

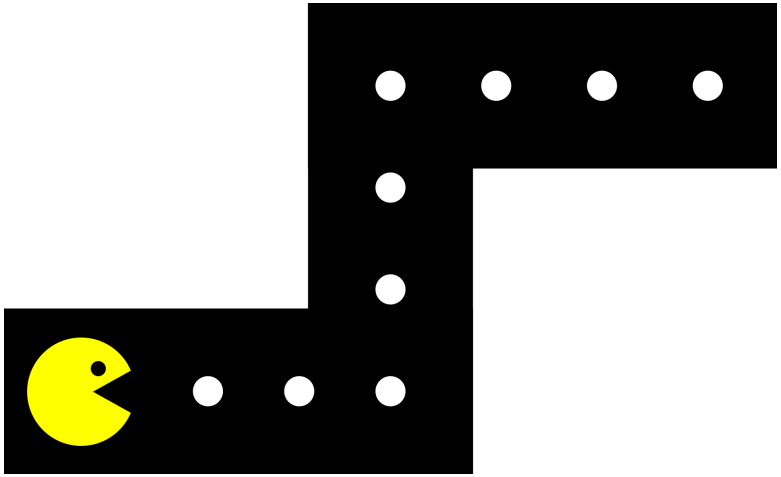
R. Chrétien, X. Baumans & B. Baert
rchretien@ulg.ac.be - xavier.baumans@ulg.ac.be

Qu'est-ce que la programmation ?

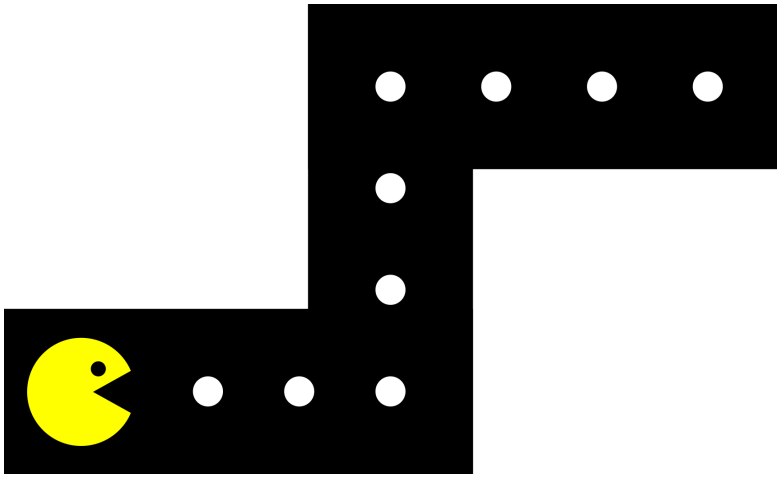
Programmer

Ecrire un programme informatique, c'est établir les étapes précises qu'un ordinateur devra exécuter pour faire quelque chose et les écrire au moyen d'un langage de programmation.

L'ordinateur exécute les étapes qu'on lui donne, une par une, et il ne peut rien faire d'autre que ce qu'on lui dit.



Qu'est-ce que la programmation ?



Exemple: diriger PacMan dans un labyrinthe

1. Avancer de trois points
2. Tourner à gauche de 90 degrés
3. Avancer de trois points
4. Tourner à droite de 90 degrés
5. Avancer de trois points

Qu'est-ce que la programmation ?

L'ordinateur n'est pas "intelligent", il est **rapide** !
Il ne comprend pas ce qu'il fait ni pourquoi il le fait: il fait ce qu'on lui dit et rien d'autre.
$\longrightarrow$ Extrême rigueur dans les instructions qu'on lui donne !

- Si on oublie une étape, l'ordinateur ne l'ajoutera jamais lui-même;
- Si une étape contient une erreur, si petite soit-elle, elle sera commise par l'ordinateur.

Même une simple faute de frappe peut conduire à une erreur dans le programme.

Objectif des TPs du cours de programmation

- Acquérir les concepts de base de la programmation informatique
- Apprendre la syntaxe d'un langage de programmation (C/C++)
- Développer des notions de logique et d'algorithmique

En pratique:

- Notions progressives et **cumulatives** lors des TPs, accompagnées d'exercices
- Très courtes **interrogations** lors de chaque séance sur la matière vue lors de la séance précédente (1 point)
- 2 interrogations plus longues en milieu et en fin de quadrimestre (3 et 4 points)

Objectif des TPs du cours de programmation

- Total de **12 points** attribués au long des interrogations, ramenés à **2 points** de la cote finale de l'examen:
  - $>$ 4 points $\rightarrow$ 1 point sur 20 à l'examen
  - $>$ 8 points $\rightarrow$ 2 points sur 20 à l'examen
- Pour réussir le cours (TPs = 70% de l'examen)
  - Faire de nombreux **exercices**
  - Travailler **régulièrement**
  - Etudier juste avant l'examen **ne fonctionnera pas**

Comment construire un programme ?

L'ordinateur n'est capable d'exécuter qu'un nombre limité d'opérations différentes
→ ce sont les **instructions fondamentales** du processeur
⟶ on réalise des instructions plus complexes en les combinant

```
1  mov     -0x18(%rax),%rax
2  mov     0x603170(%rax),%rbx
3  test    %rbx,%rbx
4  je      401c63 <main+0x173>
5  mov     $0x603080,%edi
6  mov     $0xfffe7960,%ebx
7  callq   400d08 <_ZNSo3putEc@plt>
8  mov     %rax,%rdi
```

Ce **code machine** est très difficile à lire et à comprendre pour un humain
⟶ On utilise un **langage de programmation**, plus simple à comprendre et utiliser

Langages de programmation: C/C++

Un langage de programmation est constitué

- d'un ensemble de **mots-clés** qui correspondent à des instructions (`if, else, while, ...`)
- de règles pour combiner les éléments du langage: la **syntaxe**
- d'**identifiants** pour les variables, les fonctions, etc...

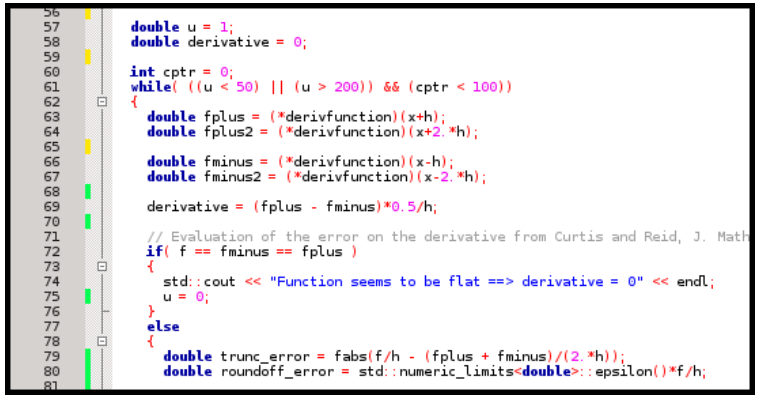
Nous utiliserons le langage **C/C++**, qui possède une syntaxe et des mots-clés qui lui sont propres

```
1 double truncation_error = fabs(f/h - (fplus +
      fminus)/(2.*h));
2 double roundoff_error =
      std::numeric_limits<double>::epsilon()*f/h;
3 u = truncation_error / roundoff_error;
```

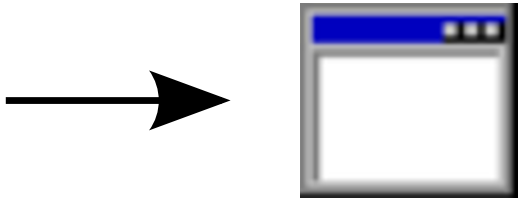
Etapes fondamentales de la construction d'un programme

2 étapes principales:

1. **Ecrire** le "code source" du programme dans un fichier texte
   → Un ou des fichier(s) texte qui contiennent le programme écrit dans un langage de programmation
2. "**Compiler**" le code source pour en faire un programme exécutable
   → Le compilateur transforme les fichiers sources en un code exécutable par l'ordinateur



```
56
57    double u = 1;
58    double derivative = 0;
59
60    int cptr = 0;
61    while( ((u < 50) || (u > 200)) && (cptr < 100))
62    {
63      double fplus = (*derivfunction)(x+h);
64      double fplus2 = (*derivfunction)(x+2.*h);
65
66      double fminus = (*derivfunction)(x-h);
67      double fminus2 = (*derivfunction)(x-2.*h);
68
69      derivative = (fplus - fminus)*0.5/h;
70
71      // Evaluation of the error on the derivative from Curtis and Reid, J. Math
72      if( f == fminus == fplus )
73      {
74        std::cout << "Function seems to be flat ==> derivative = 0" << endl;
75        u = 0;
76      }
77      else
78      {
79        double trunc_error = fabs(f/h - (fplus + fminus)/(2.*h));
80        double roundoff_error = std::numeric_limits<double>::epsilon()*f/h;
81
```

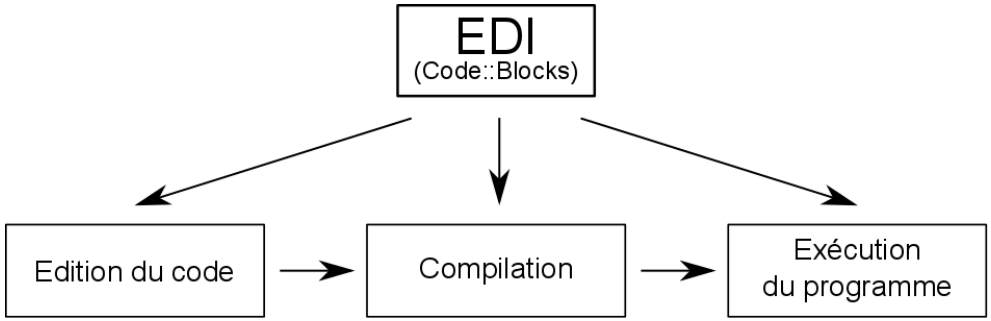


Ecrire le code source → Code::Blocks

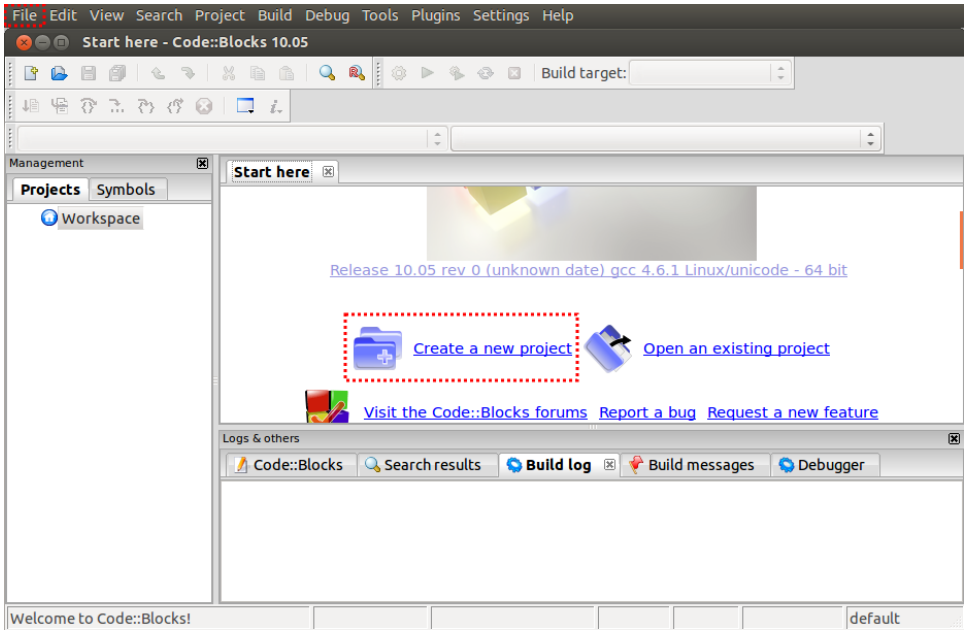
Code::Blocks = **Environnement de Développement Intégré**

Il regroupe :

- Un **éditeur de texte**: il facilite l'écriture du code source (coloration syntaxique, numérotation des lignes, indentation automatique, ...)
- Une interface avec un **compilateur**: il permet de compiler directement les codes sources écrits dans l'éditeur. Une fenêtre montre les erreurs de compilation lorsqu'il y en a. On peut ensuite exécuter le programme.

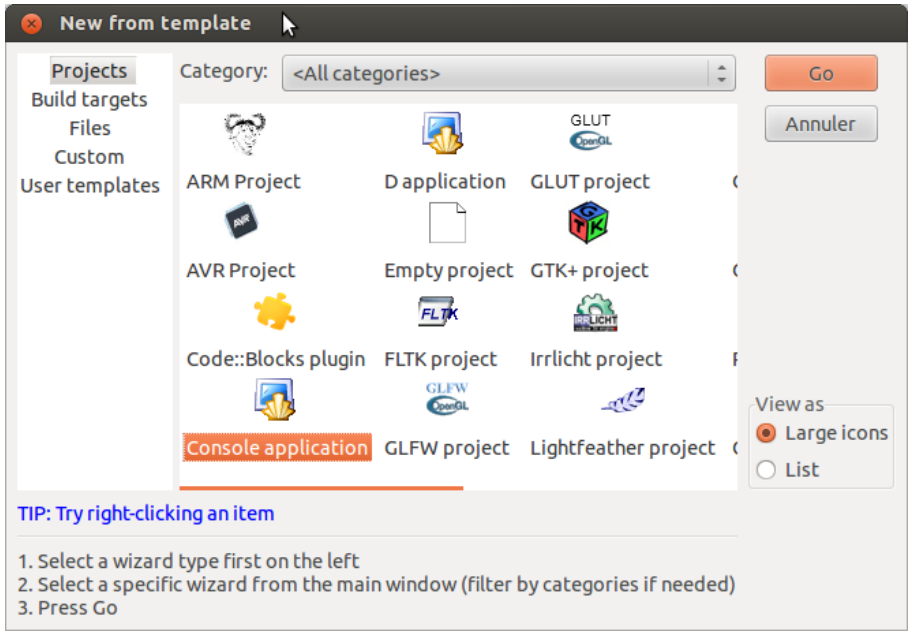


Créer un projet Code::Blocks (1/4)



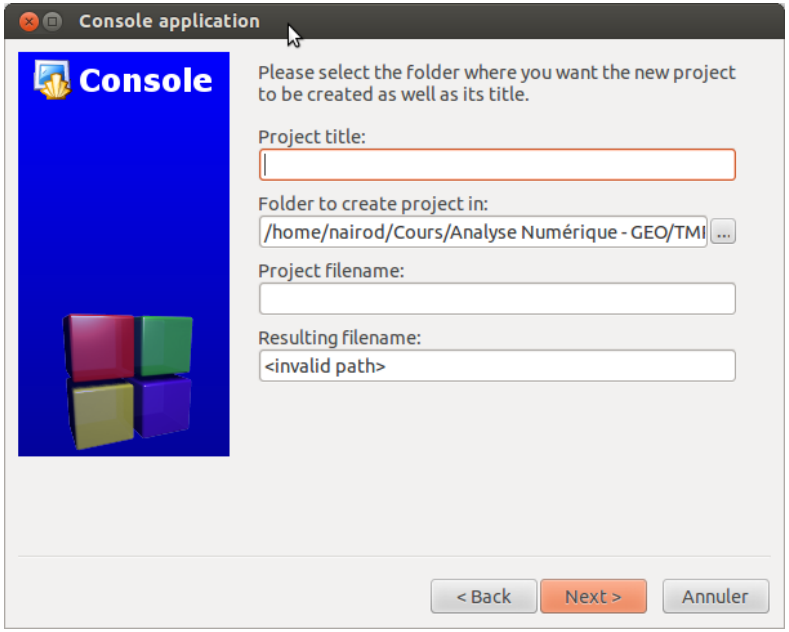
Cliquer sur "`Create a new project`"

Créer un projet Code::Blocks (2/4)



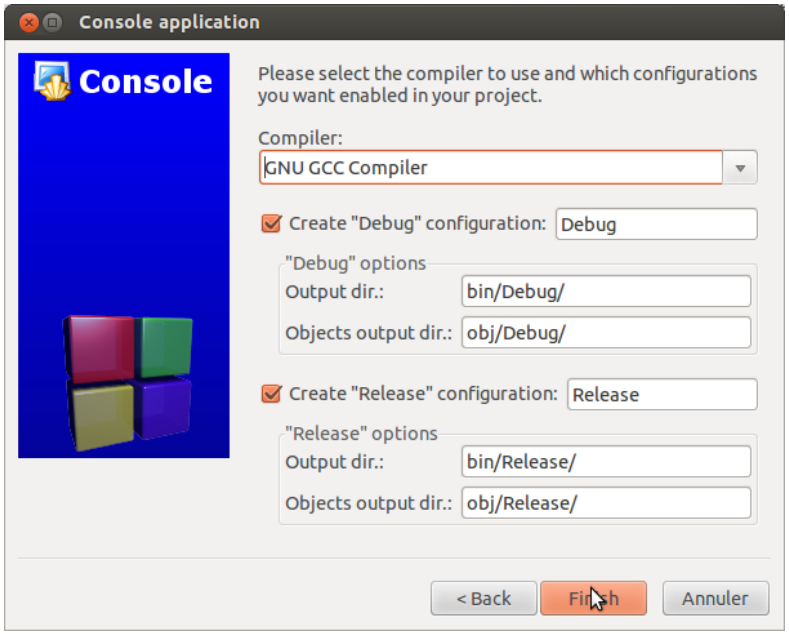
Dans la catégorie "`Projects`", choisir le type "`Console application`", pour construire un projet permettant de réaliser des affichages et des saisies au clavier dans un terminal.

Créer un projet Code::Blocks (3/4)



Choisir un nom de projet pour le champ "`Project title`" et choisir le dossier dans lequel enregistrer le projet avec le champ "`Folder to create project in`".
Laisser les valeurs par défaut pour les deux autres champs.

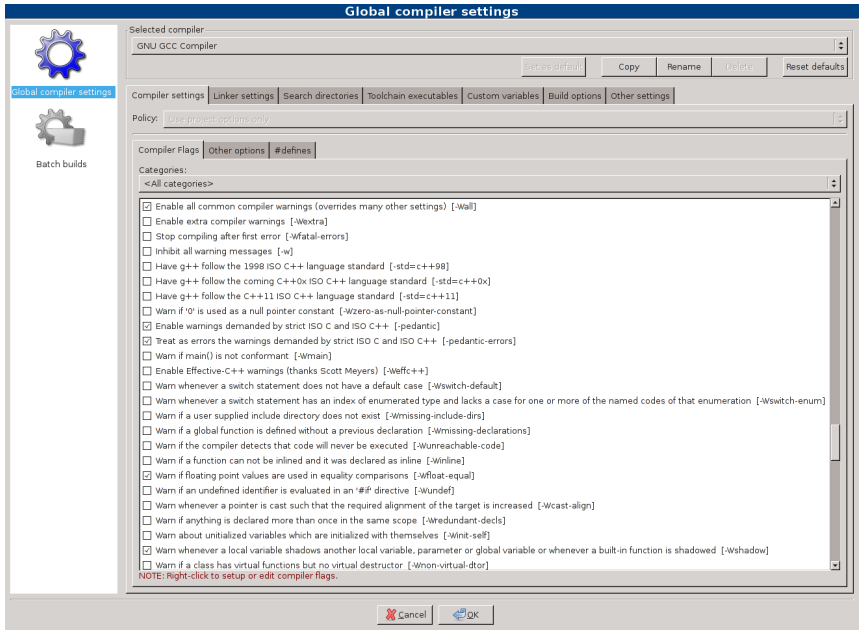
Créer un projet Code::Blocks (4/4)



Ne rien modifier dans cette fenêtre et cliquer sur "`Finish`"

Utiliser Code::Blocks chez soi (compiler flags)

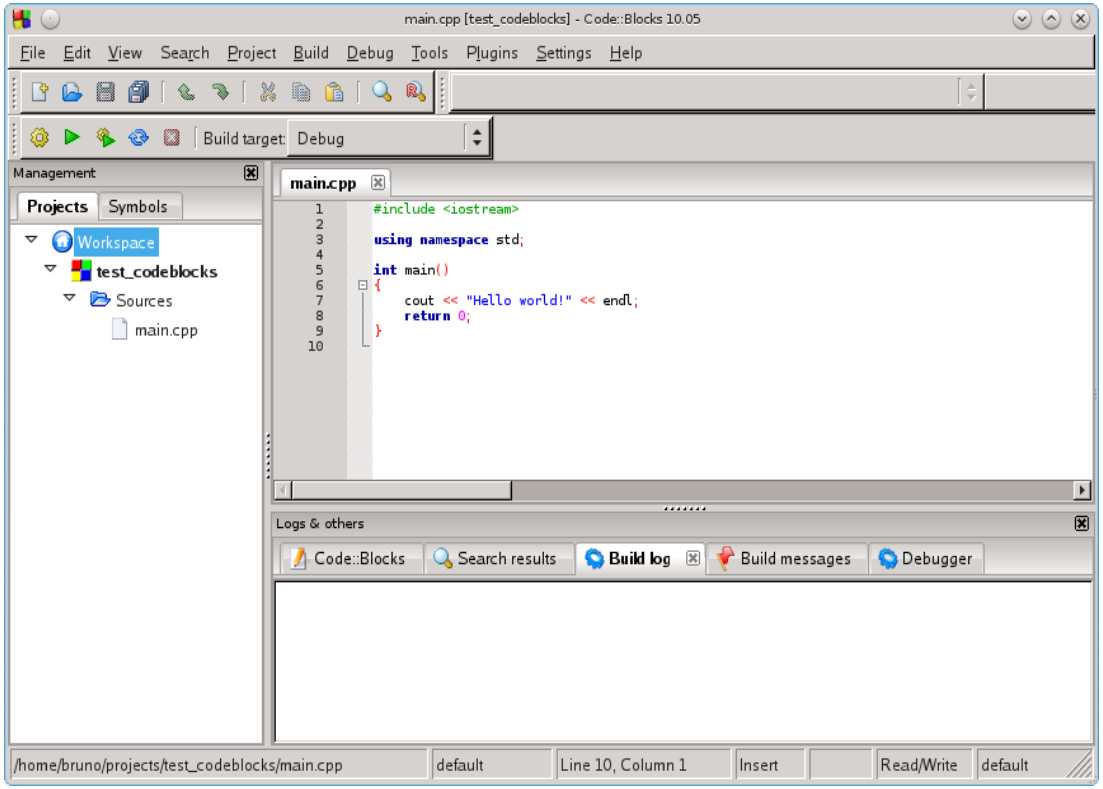
Lorsque vous utiliserez Code::Blocks chez vous, il est conseillé de modifier certains paramètres par défaut afin qu'ils correspondent à ceux utilisés lors des TPs (**et de l'examen !**)
Ceux-ci se trouvent dans le menu "Settings/Compiler..."



Les cases supplémentaires à cocher sont:

- -Wall
- -pedantic
- -pedantic-errors
- -Wfloat-equal
- -Wshadow
- -std=c++11

Premier programme: Hello World !



Lors de la création d'un nouveau projet, Code::Blocks le complète automatiquement avec un petit programme: "**Hello World !**"
Tous les programmeurs commencent leur apprentissage de la programmation en écrivant ce petit programme.

Hello World !

Le code source du programme "Hello World" en C/C++ est le suivant :

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      cout << "Hello world!" << endl;
8      return 0;
9  }
```

La partie "instructions" du programme Le programme démarre toujours en exécutant une **fonction principale** appelée "**main**" Les instructions de cette fonction sont contenues entre des **accolades** { ... } L'objet "**cout**" permet l'affichage à l'écran

Instructions en C/C++

- Les instructions sont séparées par des **points-virgules ";"**
- Les accolades {...} définissent des blocs d'instructions
  Ex. les instructions de la fonction **main()** sont entourées par des accolades
- Le nombre d'espaces entre les mots-clés, identifiants, etc... n'a pas d'importance

```
1  cout << "Hello world!";
```

```
1  cout    <<     "Hello world!";
```

Fonction main

La fonction **main()** est la fonction principale du programme.

```
1 int main()
2 {
3   // instructions du programme
4   return 0;
5 }
```

Elle doit toujours exister et elle est la première à être exécutée.

Lorsqu'elle se termine, elle donne comme résultat un nombre entier (**int**) qui permet de savoir si son exécution s'est déroulée sans problème (valeur 0) ou si des erreurs se sont produites (valeur positive).

Affichage dans la console: cout (1/2)

"**cout**" permet d'afficher des informations sur l'écran.
Les informations à afficher sont juxtaposées l'une derrière l'autre et séparées par les caractères <<.

```
1 cout << "Du texte" << " qui s'affiche" << endl;
```

L'identifiant "**endl**", lorsqu'il est passé à l'objet **cout**, provoque un retour à la ligne.

```
1 cout << "La ligne 1" << " qui s'affiche" << endl;
2 cout << "La ligne 2" << " s'affiche aussi" << endl;
```

Le caractère spécial "**t**" permet d'insérer une tabulation dans l'affichage.

```
1 cout << "Texte 1\tTexte 2\tTexte 3" << endl;
```

Affichage dans la console: cout (2/2)

On peut également afficher des nombres

```
1 cout << "Le nombre " << 361 << " s'affiche" << endl;
2 cout << "Pi: " << 3.14159265358979323846 26 << endl;
```

L'instruction **cout.precision(x)** permet de définir le nombre de chiffres significatifs à afficher

```
1 cout << "Le nombre " << 361 << " s'affiche" << endl;
2 // affichage: Le nombre 361 s'affiche
3 cout << "Pi: " << 3.14159265358979323846 26 << endl;
4 // affichage: Pi: 3.14159
5 cout.precision(10);
6 cout << "Pi: " << 3.14159265358979323846 26 << endl;
7 // affichage: Pi: 3.141592653
```

Notes de TPs

Site internet où trouver les notes de TP:

http://www.pqs.ulg.ac.be/
$-- >$ Education
$-- >$ Introduction à la programmation

Exercices

- Construire un projet "Hello World !", le compiler et l'exécuter;
- Construire un programme affichant les premières décimales du nombre d'or $\Phi = 1.6180339887498948482045 86...$, avec successivement
  - la précision par défaut de **cout**;
  - 15 chiffres significatifs;
  - 10 chiffres significatifs.
- Construire un programme affichant à l'écran un dessin de votre choix à l'aide de caractères tels que *, +, _, -, etc...

```
^__^
(oo)\_______
(__)\       )\/\
   ||----w |
   ||     ||
```

Langage de programmation visuel: Scratch

https://scratch.mit.edu/