INFO0202

Méthodes de la programmation appliquées à la chimie

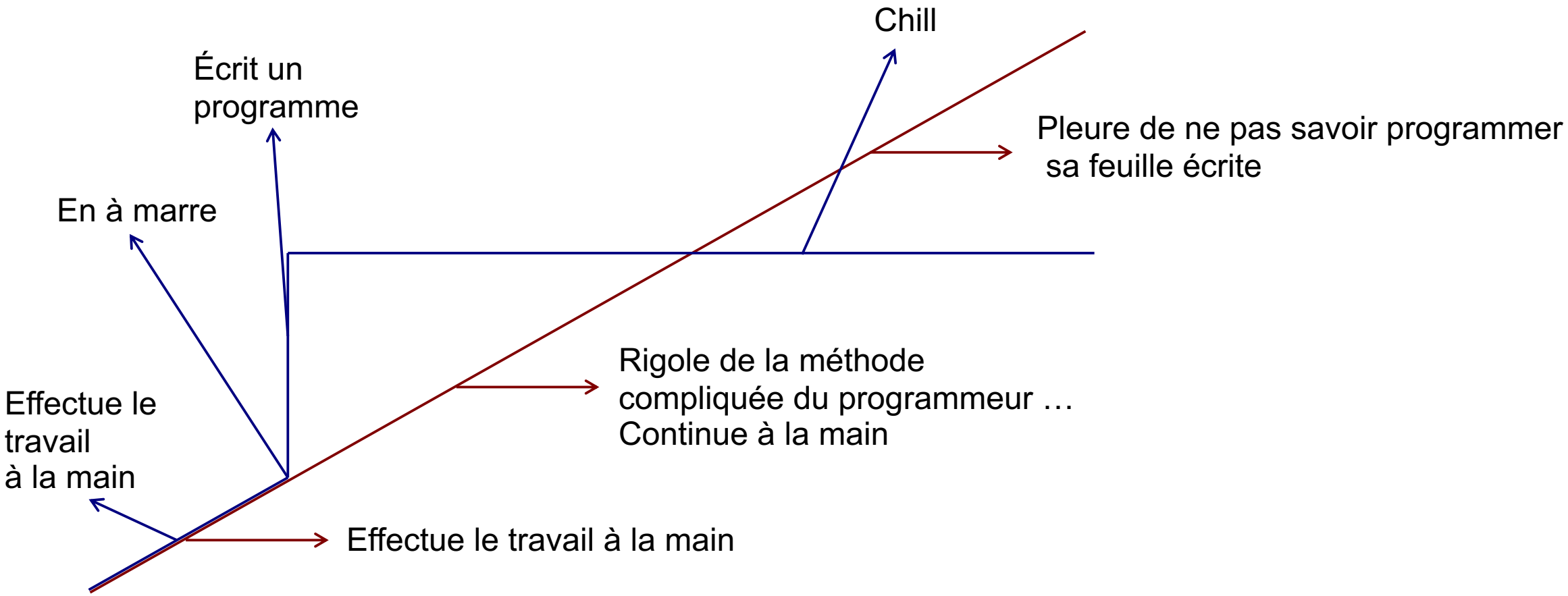
03 mars 2021

Ratz Thomas
thomas.ratz@uliege.be
Bât. B5a R54

Pourquoi programmer ?

Temps

Programmeur VS Utilisateur lambda



Taille du travail

Quoi et comment ?

Quoi ?

Définition d'une racine carré

Mathématiquement une racine carrés est définie comme:

$$\sqrt{x} = y \iff y^2 = x \,\&\, y > 0$$

Comment ?

Algorithme = implémentation numérique de la résolution d'un problème mathématique

Algorithme de Héron:

$$G_0 = x$$

$$G_{n+1} = \frac{1}{2}\left(G_n + \frac{x}{G_n}\right)$$

Tant que $|G_n - G_{n-1}| > \epsilon$

Algorithme

- Suite d'instructions, procédure

- A charge du programmeur:

  - ✓ Vérifier que l'algorithme est correct → justesse
  - ✓ Vérifier que ce dernier fonctionne dans toutes les situations → robustesse
  - ✓ Fournir l'ensemble des informations nécessaires à son bon fonctionnement

- L'ordinateur se contente d'exécuter séquentiellement (de manière successive). Elle ne réfléchit pas à la place du programmeur.

Le langage C/C++

- Langage de programmation utilisé: C/C++
  - ✓ Structure des programmes proches du C (https://waytolearnx.com/2018/12/differences-entre-c-et-c.html)
  - ✓ On utilise un compilateur C++
- Plus souple, moins strict
- Utilisation des librairies du C++ (iostream, vector,… etc)
- Utilisation du type boolén (pas intrinsèque en C)
- On n'utilisera pas cependant des fonctions avancées du C++ comme:
  - ✓ Programmation orientée objet (POO)
  - ✓ Héritage
  - ✓ …

IDE : Code::Blocks & GCC

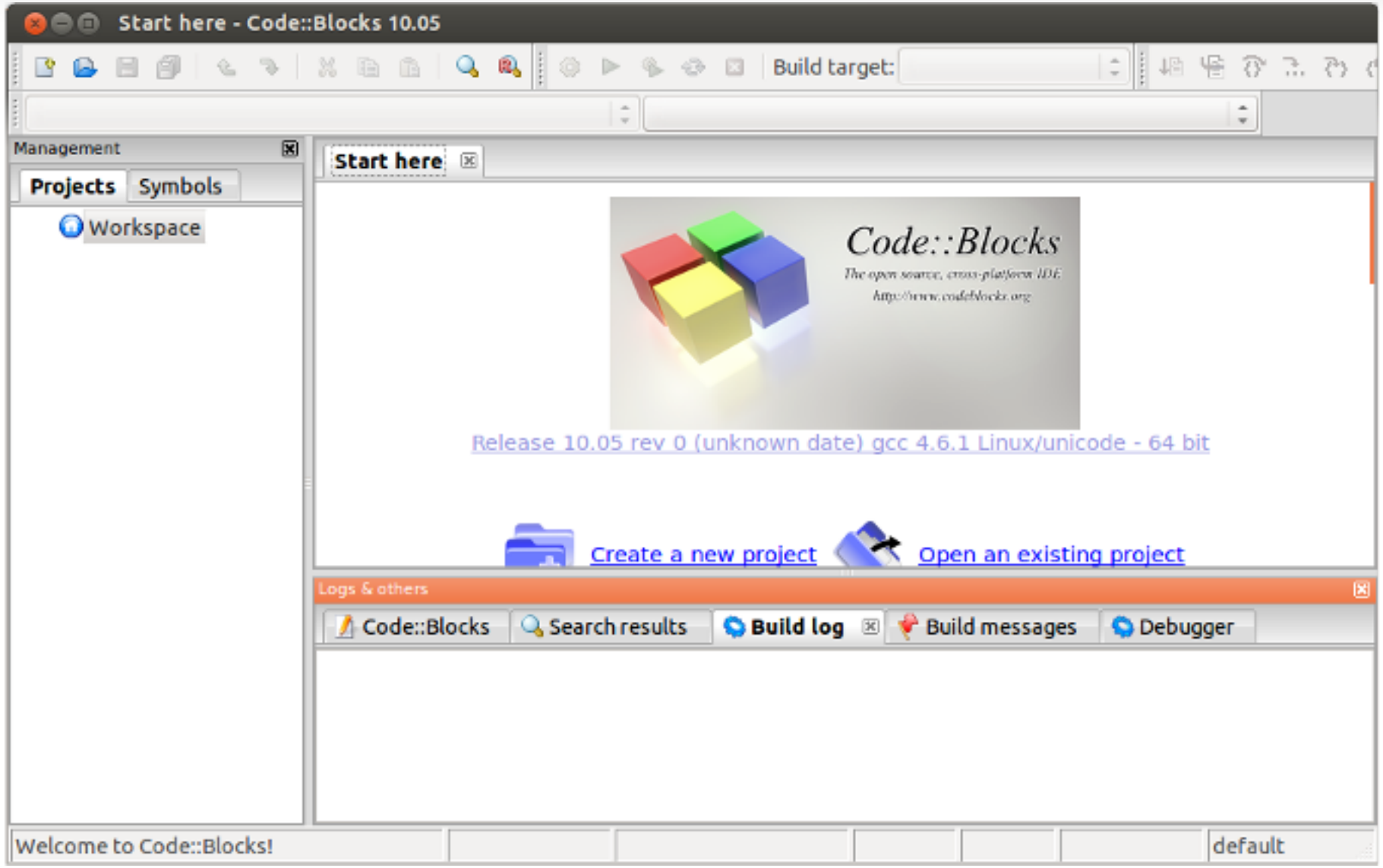
- Le code source est un simple fichier texte sans mise en forme (format .txt) qui pourrait être écrit sur bloc-note ou n'importe quelle éditeur de texte.

- Un **EDI** (Environnement de développement intégré ou IDE en anglais) est un éditeur de texte spécialisé en un/plusieurs language(s) de programmation. Ce dernier permet d'augmenter l'éfficacité de programmation.

- GCC pour "Gnu compiler collection" est une collection de compilateurs utilisée pour différents langages de programmations.

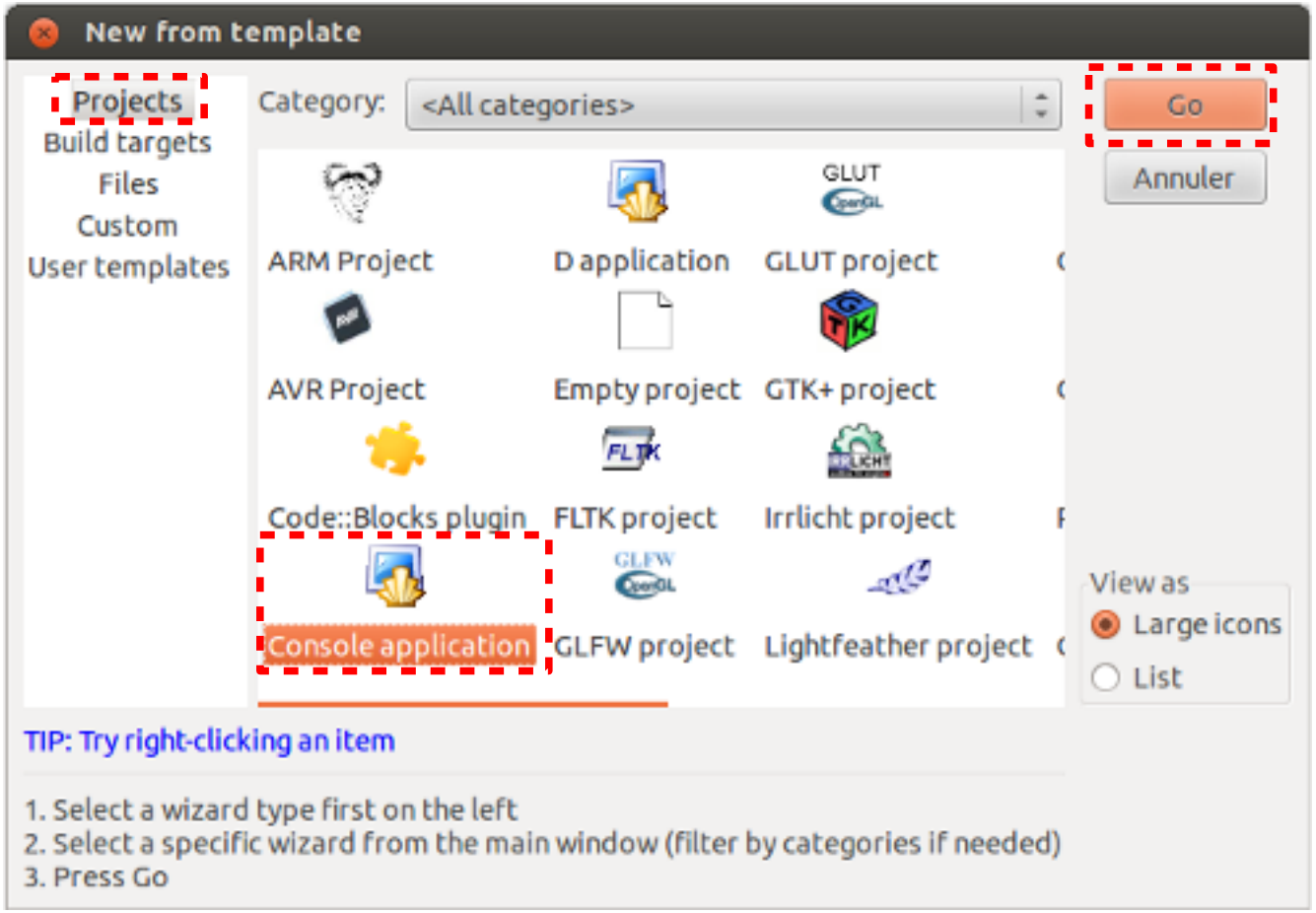
- L'**EDI** fourni:
  - ✓ Une gestion de projets (plusieurs fichiers par projet)
  - ✓ Coloration syntaxique
  - ✓ Aide à la saisie intelligente
  - ✓ Raccourci vers le compilateur

IDE : Code::Blocks & GCC



C::B Nouveau projet

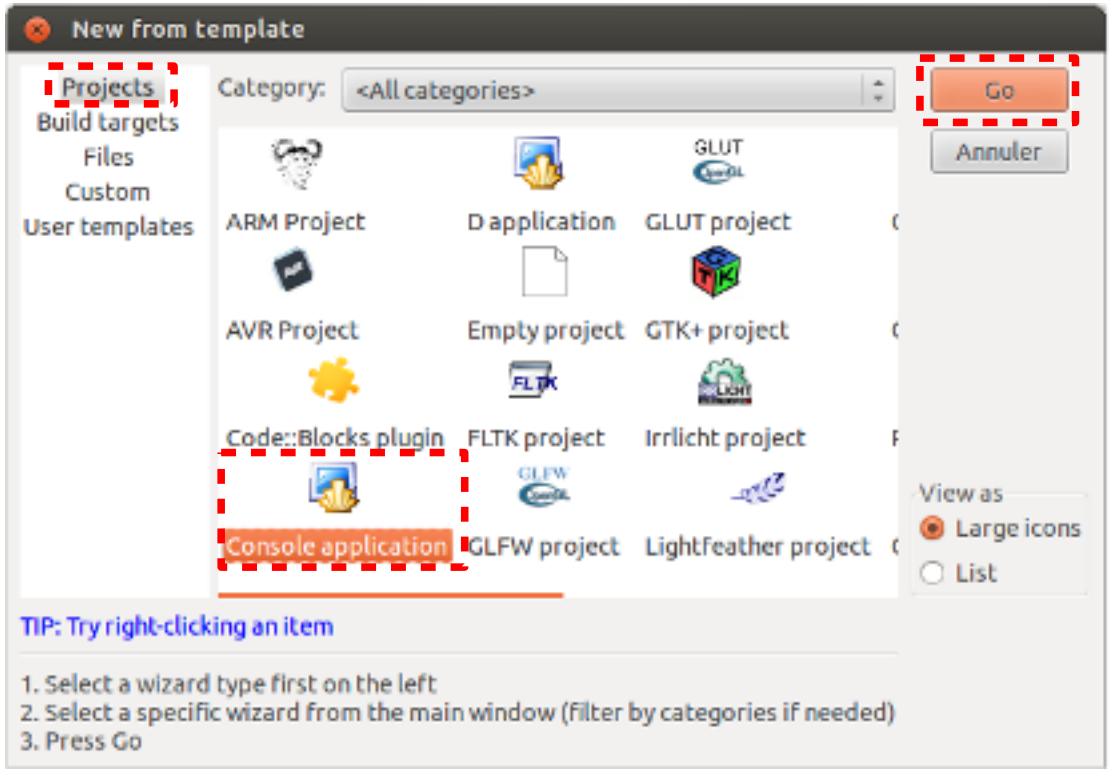
Création d'un projet = ensemble de fichiers

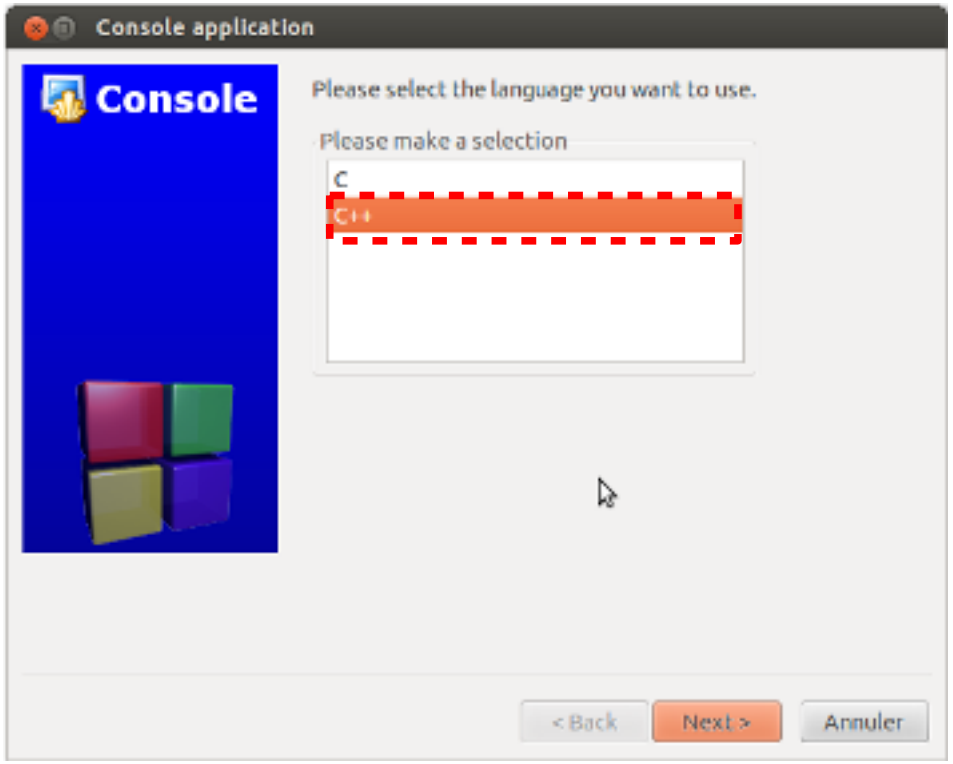


File/New/Project

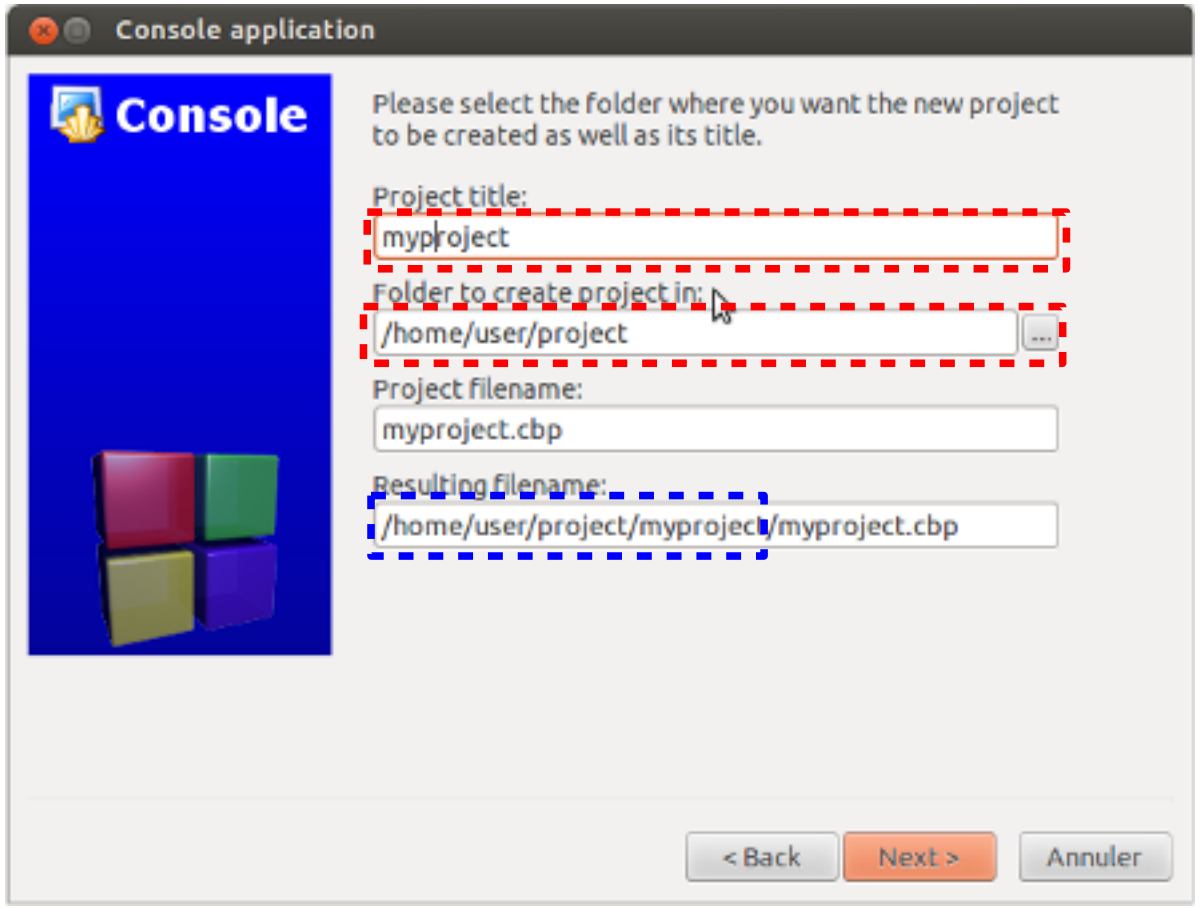
C::B Nouveau projet

Plusieurs types de projets existent, lors de ce TP, nous n'utiliserons que des **applications en mode console**.





C::B Nouveau projet



Titre du projet, nom du programme.
Un dossier avec ce nom est créé !

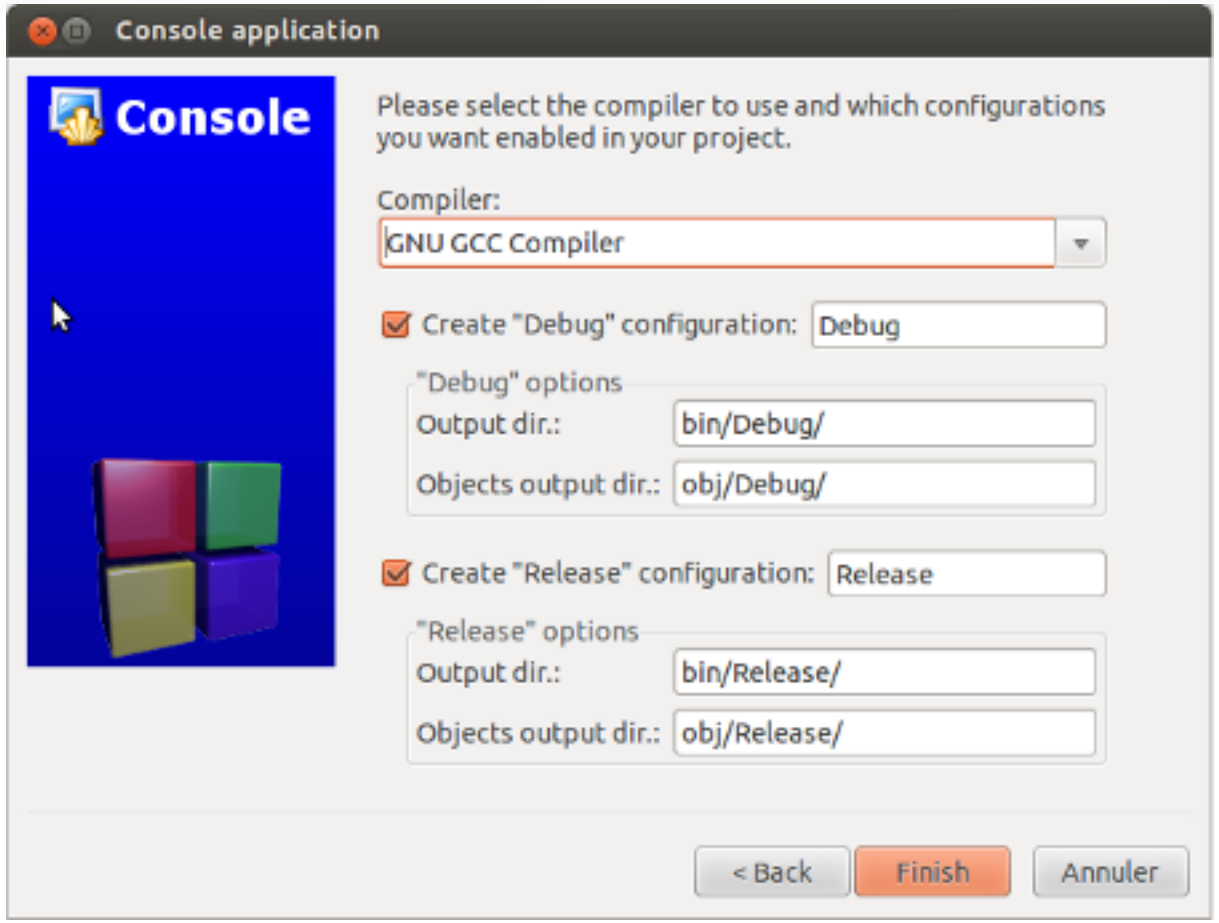
Dossier où est créé le projet. Tous les fichiers doivent se trouver là !

Nom du fichier projet. Usuellement, le même que celui du projet !

Chemin complet du fichier projet.

Dossier du projet.

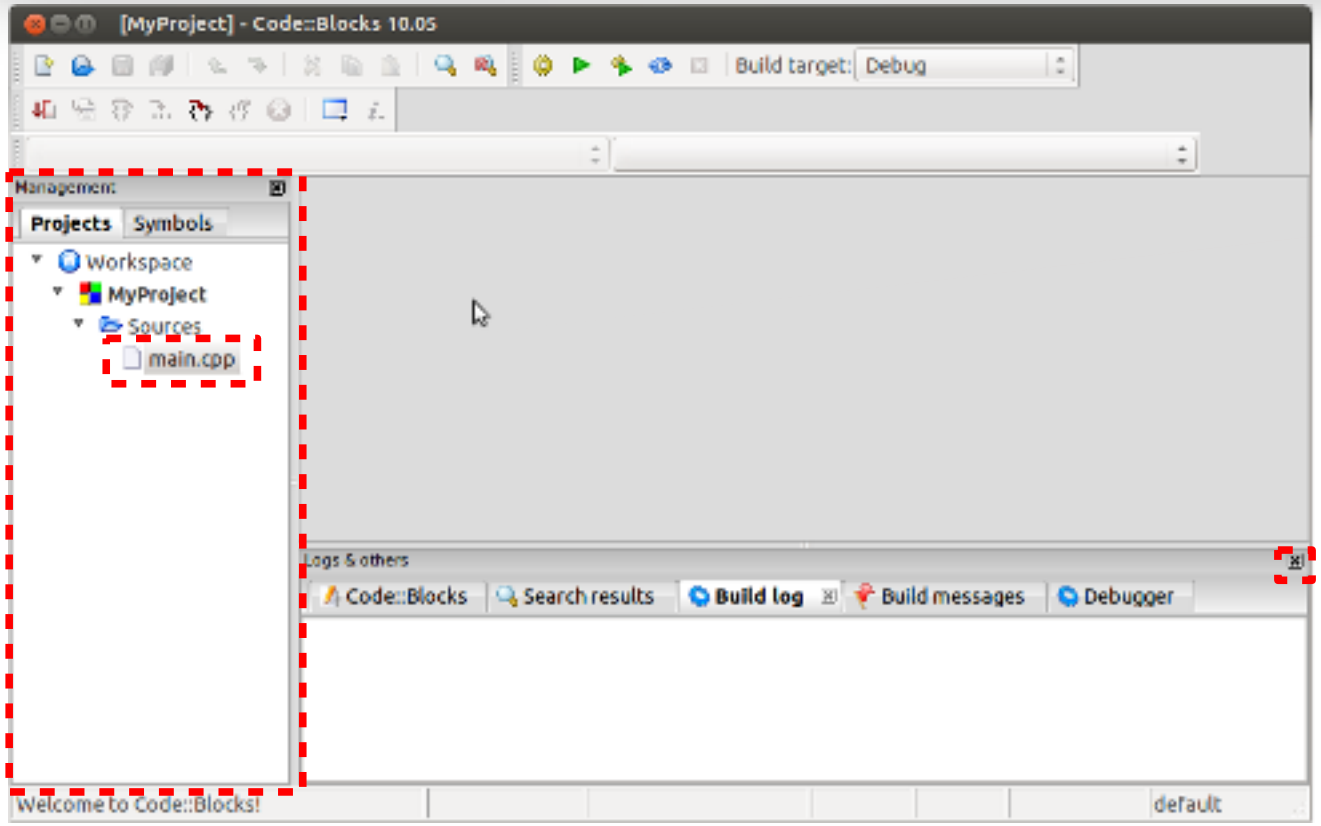
Pourquoi programmer ?



Le fichier exécutable final créé sera situé dans le sous-dossier *Debug/Release* du dossier *bin* du *dossier projet* définit précédemment !

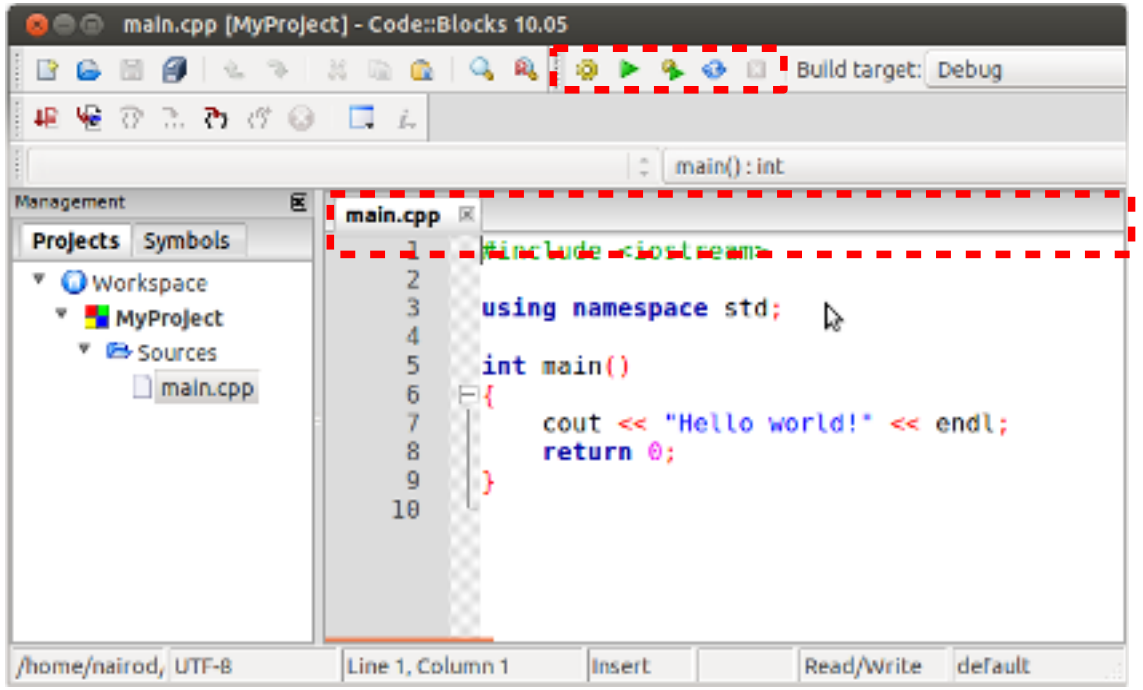
Cette fenêtre donne les configurations de compilation/debug/release du projet, il n'y a rien à modifier.

Nouveau projet



Tous les fichiers actuellement dans le projet se trouvent dans le répertoire virtuel ***Sources***, sous le nom du projet (***MyProject***) de l'onglet **Projects** de la fenêtre **Management**.

Si celle-ci n'est pas visible : View → Manager



Double cliquer sur un nom de fichier pour l'ouvrir.

L'ensemble des fichiers ouverts apparaît dans la barre d'onglet.

Compiler / Exécuter / Compiler & Exécuter / Tout recompiler / Stop

Premier programme


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

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

- Une instruction se termine toujours par ;
- Les {} délimitent des blocs de codes (des parties de codes contenues dans le ”squelette” principal int main(){ … }
- **Le return 0;** marque la fin du programme
- La commande #include <...> en début de code permet d’inclure des librairies
- Enfin, **using namespace std;** Une classe standard pour l’appel de fonctions

Variables

- Les variables servent à stocker des données

- Elles possèdent un **nom** (afin de s'y référer), une **adresse** correspondant à où la variable est stockée dans l'ordinateur et un **type** (nature des données). Parmi ces types, il y a:

  - ✓ Des nombres entiers (**int**, **long**, …)
  - ✓ Des nombres réels (**float, double**, …)
  - ✓ Des booléens (**bool**)
  - ✓ Des caractères (**char**)
  - ✓ Des ensembles de caractères (**string**)
  - ✓ …

- Le nom de la variable doit être choisi avec soin !

Type des variables

Entier	int	long	short	unsigned int
Flottant	double	float		
Caractère	char	*wchar*		
Booléen	bool			

```
...
int main()
{
char myCaractere;
myCaractere = 'c';

int a;
double myDouble = 4.5;

   return 0;
}
```

Différents types de varaibles avec différentes tailles et précisions:

type de variable	Signification	Octets	Plage de valeurs acceptées
char	Caractère	1	0 à 255
int	Entier	2	-32 768 à 32 767
long int	Entier long	8	-2 147 483 648 à 2 147 483 647
float	Flottant	4	$3,4\ 10^{-38}$ à $3,4\ 10^{38}$ env.
double	Flottant double	8	$1,7\ 10^{-308}$ à $1,7\ 10^{308}$ env.

Constantes

- Les constantes permettent de stocker des valeurs typées sans risque de modification

- Elles sont généralement écrites en majuscules

```
const double PI = 3.141592;

int main()
{
   const int SIZEMAX = 5000;

   PI = 3.14; // Erreur !

   return 0;
}
```

Les opérateurs

Les opérateurs nécessitents un ou deux éléments (opérandes) sur lequel/lesquels appliquer l'opértateur:

- ✓ Arithmétique: +, -, *, %
- ✓ Affectation: =, +=, -=, *=, /=, %=, ++, --
- ✓ Comparaison >, >=, ==, <=, <, !=
- ✓ Logique && (et), || (ou), ! (non)

Attention à la différence entre affecter et compare !
"a=b" signifie que le contenu de la variable b est placée dans le'espace mémoire représenté par la variable a

```
int a = 7, b = 5, c = 3, inbr;
double pi = 3.14159, e = 2.71828, phi = 1.6180, dnbr;
bool result;

dnbr = (pi+phi)*e;          // dnbr vaut 12.93789
inbr = a%b-c;               // inbr vaut -1
inbr += b-c*c;              // inbr vaut -5
result = !((b > a) || (b > c));// result vaut false/0
```

Les changements de type

Il est parfois nécessaire de changer le type d'une variable. Par exemple lors d'une opération de division.

```
...
int a = 27, b = 5, c = 3, inbr, bigInt = 2000000000;
double pi = 3.14159, e = 2.71828, phi = 1.6180, dnbr;
bool result;

dnbr = a/b;                  //DIVISION ENTIERE 27/5=5

dnbr = (double)a/(double)b; //DIVISION RATIONNELLE =5.4
result = (bool)(b-5);
result = (bool)(c-5);

short littleInt = (short)bigInt;  // PERTE DE DONNEE
...
```

En cas de doute, il vaut mieux forcer le type à utiliser mais attention à la perte de données !

Affichage à l'écran

- L'objet **cout** permet d'afficher des variables et du texte dans la console (librairie iostream)
- L'opérateur << signifie sortie (à gauche)
- L'élément **endl** provoque un retour à la ligne tandis que '**\t** ' est une tabulation
- Chaque élément à afficher de type différent doit être séparé

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

int main()
{
int a = 5;
double d = 4.5;
cout << "La valeur de a est" << '\t' << a << endl;
cout << "La valeur du double est" << d << endl;
return 0;
}
```

Paramètre d'affichage

- Il est possible de formater l'affichage des nombres réels via la librairie iomanip

  - ✓ Fixer la précision
  - ✓ Choisir l'affichage scientifique/fixe
  - ✓ Choisir la base (oct/dec/hex)
  - ✓ Fixer la largeur nécessaire (alignement)
  - ✓ Afficher true/false plutôt que 1/0
  - ✓ …

http://www.cplusplus.com/reference/iostream/manipulators/

Saisie de données au clavier

- L'objet **cin** permet de lire des données tapées dans la console par l'utilisateur et de stocker la valeur dans un espace mémoire associé à une variable.

- L'opérateur >> signifie entrée (vers le droite)

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

int main()
{
int a;
cout << "Entrez un nombre entier : ";
cin >> a;
cout << endl << "Le carre de votre nombre est ";
cout << a*a << endl;
return 0;
}
```

Cout et cin

- Deux objets (C++) de la librairie iostream qui appartiennent à l'espace de nom standard (std)
- Les échanges se font par flux (stream)
- Ils correspondent à l'entrée et la sortie standards
  - ✓ Entrée au clavier (PC)
  - ✓ Flux réseau (serveur)
- L'objet cin est bloquant
  - ✓ Appuyer sur ENTER pour terminer l'acquisiation des données
  - ✓ On peut vider le buffer avec **cin**.ignore()
  - ✓ On peut récupérer un seul caractère avec **cin**.get()

Voir exemple 1

Commentaires

Les commentaires sont ignorés par le compilateur et servent à documenter le code

```
#include <iostream> //inclure une librairie
using namespace std;

int main() //début du corps du code
{
int a;
cout << "Entrez un nombre entier : "; //Demande
cin >> a;
/*cout << endl << "Le carre de votre nombre est ";
cout << a*a << endl;*/
// Tout ceci est ignoré
return 0;
}
```

Ajouter des commentaires permet d'agmenter la lisibilité et la compréhension du code !

Indentation

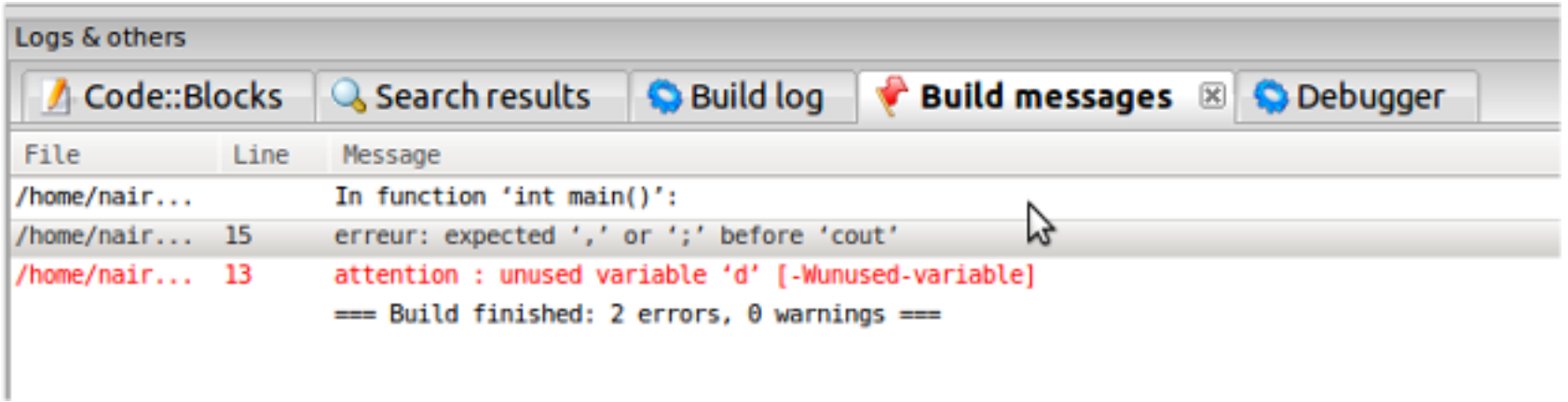
Il existe des convention en terme d'écriture de codes !

```
void a_function(void)
{
    if (x == y) {
        something1();
        something2();
    } else {
        somethingelse1();
        somethingelse2();
    }
    finalthing();
}
```

```
void a_function (void)
{
if (x == y)
      {
      something1 ();
      something2 ();
      }
      else
   {
      somethingelse1 ();
      somethingelse2 ();
   }
      finalthing ();
}
```

Les bugs

- Deux grand types d'erreurs lors de l'écriture d'un programme:
  - ✓ Les erreurs de syntaxe (compilation)
  - ✓ Les erreurs logiques (exécution)
- Contrairement aux erreurs logiques, les erreurs de syntaxes sont détectées lors de la compilation:
  - ✓ Oubli de ;
  - ✓ Mauvais appariement de paranthèses (4+5*(1+3)
  - ✓ …



Les erreurs logiques

Pour éviter les erreurs logiques, il est préférable de:

- ❑ Compiler son code régulièrement de façon à détecter les erreurs le plus tôt possible
- ❑ Vérifier que les lignes de codes récemment ajoutées fournissent le résultat attendu
- ❑ Ajouter des lignes de code de vérification (**cout** de résultats intermédiaires)
- ❑ Débugger le code grâce au debugger

Règles de bonne pratique

Lors de l'écriture d'un programme, respectez les règles suivantes:

- ❑ Le programme doit être correct (justesse)
- ❑ Le programme doit fonctionner dans toute situation (robustesse)
- ❑ Il doit être lisible (bien indenté et commenté)
- ❑ Les noms de varaibles et des fonctions doivent être pertinents !
- ❑ Le programme doit être concis (plus un programme est cours moins il y a d'erreurs)
- ❑ Comprenez les lignes que vous écrivez
- ❑ Compilez votre programme régulièrement afin d'identifier des erreurs éventuelles
- ❑ Ajouter des lignes de codes de vérifications
- ❑ En cas de doutes, référez-vous à la documentation

- Cours, Exemples, Tutoriels
- *cplusplus.com* (référence complète de la STL - **eng**)
- *developpez.com* (cours, forum, autres langages - **fr**)
- *learncpp.com* (cours ultra-complet de C++ - **eng**)

Exercices

- Déclarez deux variables de type entières et deux variables de type réels. Affectez ces variables à des valeurs saisies au clavier et afficher le contenu de ces variables dans la console. Effectuez les opérations ‘+’,’-’,’*’,’/’ sur chacun des couples de variables et affichez les résultats dans la console.

- Créez un programme qui affiche dans la console la table de vérité de l’opérateur logique AND (&&)

true	&&	**true**	=	**true**
true	&&	**false**	=	**false**
false	&&	**true**	=	**false**
false	&&	**false**	=	**false**

- Déclarez deux variables de types entières auxquelles vous affecterez une valeur saisie au clavier. Echanger ensuite le contenu des deux variables et affichez dans la console le contenu des variables avant et après l’échange des valeurs. Afin de réalier cette tâche, il est nécessaire de créer une variable tampon.

Les structures de contrôle

- Exécuter / répéter des parties de code en fonction du contexte (valeur de certaines variables, …)

- Branchement conditionnel:
  - ✓ **If … else**
  - ✓ **Switch case**

- Boucles
  - ✓ **While, do … while**
  - ✓ **for**

If … else

Permet d'exécuter une partie de code si une condition est vérifiée (=**true**)

```
...
int a = 2, b = 5;

if(a > b)
cout << a << " est plus grand que " << b << endl;

    if(b > a)
    {
        cout << b << " est plus grand que " << a << endl;
    }

    if(a > b)
        cout << a << " est plus grand que " << b << endl;
else
        cout << a << " est plus petit ou égale à " << b << endl;
```

Si le code à exécuter est constitué d'une seule instruction, les crochets ne sont pas nécessaires.

Si le code à exécuter est constituté des plusisuers instructions, les crochest sont obligatoires.

Les conditions peuvent être imbriquées.

Syntaxe du if … else


```
if(expression_booléenne)
instruction;

    if(expression_booléenne)
    {
        instruction;          // s'il y a plus d'une instruction
        instruction;          // les crochets {} sont nécessaires
        …
        instruction;
    }

    if(expression_booléenne)
        instruction;
else                          // exécuté si l'évaluation rend false
    {
        instruction;
        instruction;
    }
```

Switch

Si une variable doit être comparée à plusieurs constantes ou valeur, l'instruction **switch** est plus élégante qu'un ensemble de **if**, **else if**, …

```
char myChar = 'c' ;

switch(myChar)
{
case 'a' :
cout << "1er lettre de l'alphabet" << endl;
break;

        case 'b' :
            cout << "2em lettre de l'alphabet" << endl;
        break;
        ...
        default :
            cout << "n'est pas une lettre..." << endl;
}
```

While

Un bloc de code ou une instruction est exécutée tant qu'une expression est vérifiée (=**true**)

```
...

double x = 16;       // valeur initiale
double y = x;
double eps = 0.001;  // précision souhaitée

while((y*y-x) > eps)
{
y = 0.5*(y+x/y);
}
```

On utilise une boucle de type **while** quand on ne sait pas combien de fois le bloc de code doit se répéter !

For

Un bloc de code ou une instruction est exécuté tant qu'une expression est vérifiée (=**true**)
Il y a dans les paramètres de la boucle une expression d'initialisation et une de mise à jour.

```
int x = 0;

for(int i = 1; i <= 10; i++)     //La variable i n'est
{                                      //définie que dans la boucle
    x += i ;                           //calcule la somme 1+2+...+10
}
```

On utilise une boucle de type **for** quand on sait indicer chaque itération ! (ex : parcourir un tableau)

Syntaxe du for


```
for(init; expression_booléenne; mise_à_jour)
{
        instruction;
        ...
        instruction;
}
```

init n'est exécuté qu'une seule fois **avant** toutes les instructions
expression_booléenne est vérifié **avant** chaque itération (y compris la première)
mise_à_jour est exécuté **après** chaque itération

Les boucles **for** et **while** sont interchangeables !

Voir exemple 2

Exercices

- Créez un programme permettant de sommer les n premiers multiples d'un nombre k. Par exemple, la somme des 6 premiers multiples de 3 est 3 + 6 + 9 + 12 + 15 + 18 = 63. Le programme demandera à l'utilisateur d'entrer les deux nombres entiers n et k.

- La suite de Fibonacci 1, 1, 2, 3, 5, 8, 13, 21, ... est une suite de nombres entiers dont chaque termes est la somme des deux termes qui le précédent. Le rapport de deux termes consécutifs de cette suite tend vers le nombre d'or ϕ. Créez un programme qui demande à l'utilisateur d'entrer une précision et estime le nombre d'or à cette précision. Dans ce but, calculez les éléments de la suite de Fibonacci jusqu'à ce que la différence entre les valeurs de ϕ calculées à deux étapes consécutives soit inférieur à la précision souhaitée. La fonction valeur absolue (fabs) est définie dans la librairie cmath (qu'il ne faut pas oublier d'inclure au début du programme).

Fibonacci et le nombre d’or

a1 est initialisé à 1
a2 est initialisé à 1
a3 est initialisé à 2

phi est initialisé à a3/a2

Exécuter

Sauvegarder la valeur de phi dans oldphi

a1 devient a2

a2 devient a3

a3 devient a3+a1

phi devient a3/a2

jusqu’à ce que |oldphi − phi| < précision souhaitée

Exercices

Créez un programme qui demande à l'utilisateur d'entrer deux nombres entiers et calcule leur PGCD (Plus Grand Commun Diviseur). Si les deux nombres sont premiers entre eux – c'est-à-dire leur PGCD vaut 1 - le programme affichera « nombre1 (valeur) et nombre2 (valeur) sont premiers entre eux » et sinon il affichera leur PGCD.

Pour calculer le PGCD, on se base sur les propriétés suivantes

$$\text{PGCD}(x,y) = \begin{cases} PGCD(x-y,y) \; si \; y \leqslant x \\ PGCD(x,y-x) \; si \; x \leqslant y \end{cases}$$, *cette propriété vient du fait que si un nombre en divise deux autres, alors il divise forcément leur somme et leur différence.*

Le principe de l'algorithme est le suivant : on modifie à chaque étape *x* ou *y* de façon à obtenir un nouveau couple de valeurs (*x'*,*y'*) plus petites et toujours positives. Une fois que l'une de ces deux valeurs est nulle, la valeur non nulle restante est le PGCD

Exercices

Entrer x et y

Tant que x et y sont tous les deux différents de 0

Si y plus petit x

    on retranche y à x

Sinon

    on retranche x à

Si x vaut 0 le PGCD est y

Sinon le PGCD est x

Exercices supplémentaires

Créez un programme qui affiche dans la console les nombres entiers compris entre 1 et 1000 qui sont multiples de 5 et de 11 mais pas de 17. Donnez le nombre total de ces multiples (utilisation d'un compteur).
Créez un programme qui affiche dans la console les 100 premiers nombres entiers qui sont multiples de 5 et de 11 mais pas de 17.
Créez un programme permettant de calculer les séries

$$S_1 = \sum_{i=1}^{N} 1/i^2 \qquad S_2 = \sum_{i=1}^{N} (-1)^i/i$$

pour N donné par l'utilisateur.