Méthodes informatiques pour physiciens

introduction à C++ et
résolution de problèmes de physique par ordinateur

Leçon # 1 : Introduction

Alessandro Bravar Alessandro.Bravar@unige.ch
tél.: 96210 bureau: EP 206

assistants

Johanna Gramling Johanna.Gramling@unige.ch
tél.: 96368 bureau: EP 202A

Mark Rayner Mark.Rayner@unige.ch
tél.: 96263 bureau: EP 219

http://dpnc.unige.ch/~bravar/C++2015/L1
pour les notes du cours, les exemples, les corrigés, …

Introduction

Le but de ce cours est de vous initier
1. à la programmation
2. au calcul scientifique (p. ex. l'utilisation d'un ordinateur pour résoudre des problèmes)

Le langage de programmation choisi est C++. Aujourd'hui, c'est aussi le plus répandu dans le monde scientifique. Les systèmes d'exploitation (Unix, Linux, Windows…) sont tous écrits en C ou C++. P.ex. Matlab ou Maple (langage utilisateur) sont écrit en C++.

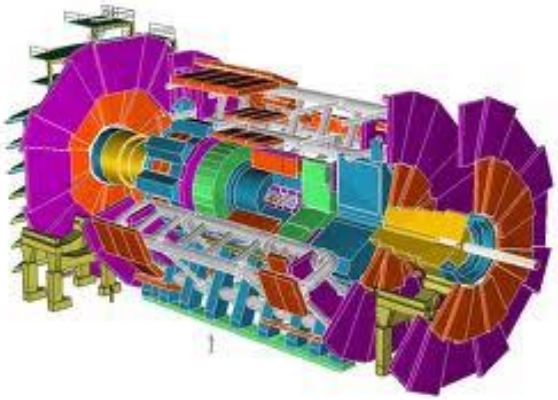
Le cours consiste en une introduction au langage de programmation C++ :
- acquisition des connaissances syntaxiques du langage pour pouvoir disposer rapidement des bases du langage et écrire des programmes
- programmation structurée et algorithmes
- programmation orientée objet

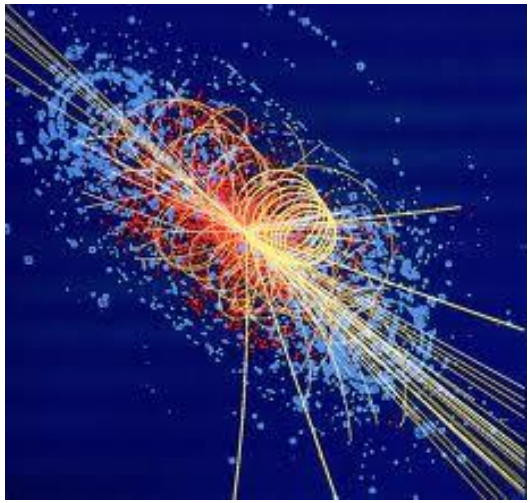
Le but est de vous mettre en condition d'écrire des programmes le plus tôt possible et de les utiliser pour résoudre des problèmes de physique réels.
L'organisation du cours (sujets) est un peu différent de cela d'un cours C++ classique.

Nous étudierons des méthodes numériques simples, mais efficaces, permettant de résoudre avec un ordinateur des problèmes de physique qui ne peuvent pas être traités simplement (i. e. différentiation et intégration numériques, résolution numérique des équations différentielles ordinaires).
Les problèmes sont basés sur la physique étudiée en première année.

Aucune connaissance préalable de programmation est requise.

Pour développer et faire voler cet avion il faut plusieurs millions de lignes de code :
- design et aérodynamique de l'avion
- système embarqué «fly by wire»
- système de navigation
- …..





Pour analyser les données de cette expérience (ATLAS @ CERN) on utilise plusieurs millions de lignes de code :
- simulations
- acquisition des données
- reconstruction
- analyse
- …..

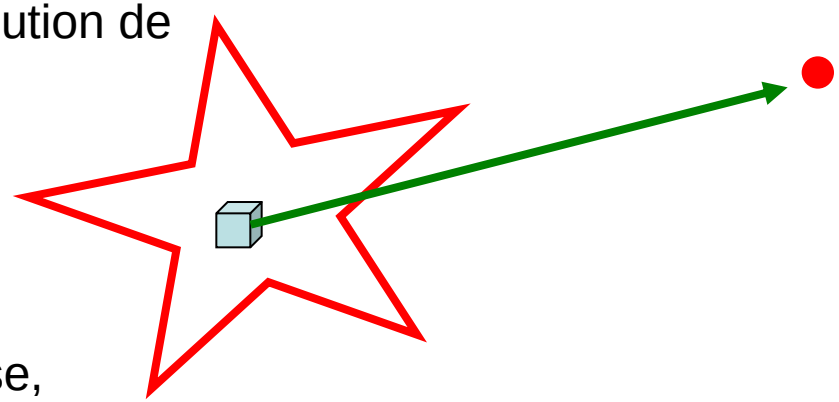
marché financière :
- analyse du marché
- prévisions
(eq. différentielles stochastiques)
- investissements
- …..

météo et climat :
- analyse des données
- modélisation
- prévisions
- …..

Quelques exemples d'intégration numérique

Champ gravitationnel généré par une distribution de masse quelconque :

$$G(r) = g_N \int_V \frac{\rho(\vec{x})}{|\vec{r} - \vec{x}|} dV$$



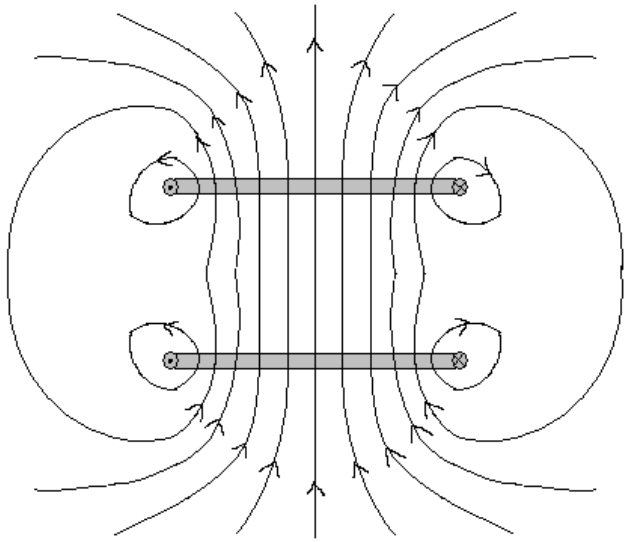
Il faut calculer l'intégrale tridimensionnelle sur tout le volume de la distribution de masse, tache plutôt simple avec des techniques d'intégration numérique, mais presque impossible analytiquement.

Champ magnétique généré par une distribution du courant (p.ex. un anneau ou solénoïde parcouru par un courant I).
On sait calculer analytiquement **B** sur l'axe de symétrie, mais par dans un point quelconque.

Pour calculer **B** dans un point quelconque, il faut intégrer la loi de Biot-Savart

$$\mathbf{B}(x) = \frac{\mu_0}{4\pi} I \int_l \frac{d\mathbf{l} \times \mathbf{r}}{r^3}$$

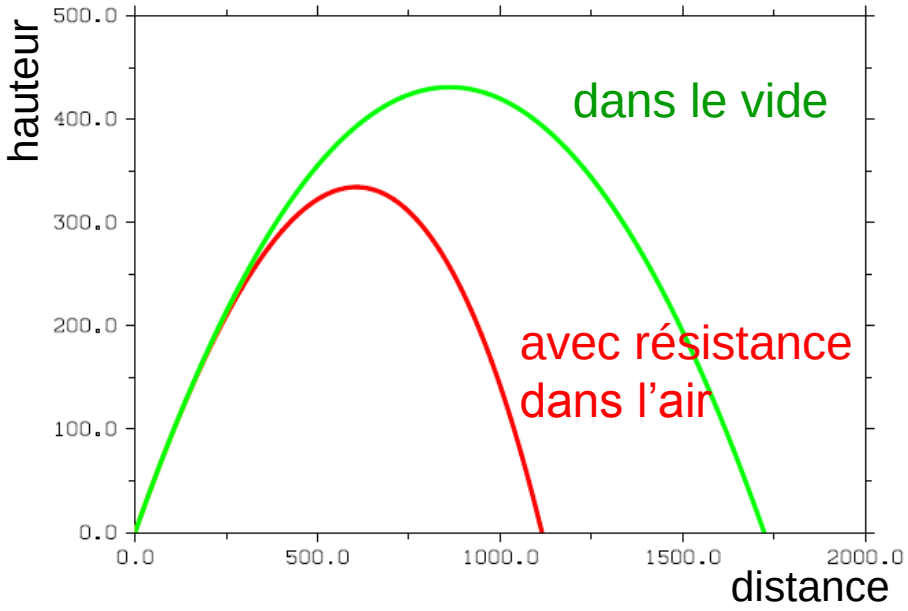
sur tout les courants.



bobines de Helmholtz

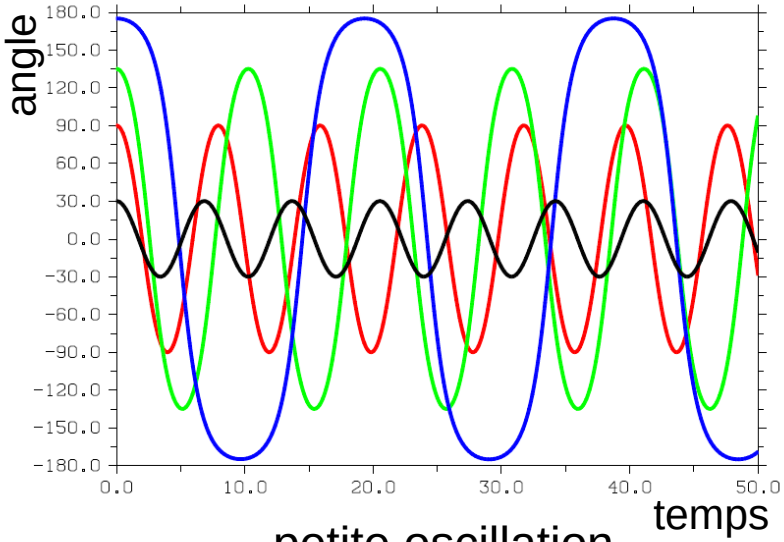
Quelques exemples d' équations différentielles

Pour décrire correctement la trajectoire d'un projectile, il faut tenir compte de la résistance de l'air (on ne fait pas de la physique dans le vide !). La trajectoire est modifiée par le frottement, ainsi l'angle du jeté maximal n'est plus de 45^0.



L' oscillateur harmonique est une très bonne approximation du pendule physique pour des petits angles. Pour des angles importants, l'approximation du petit angle n'est plus valable et il faut résoudre l'équation exacte du mouvement (la période dépend de θ !) :

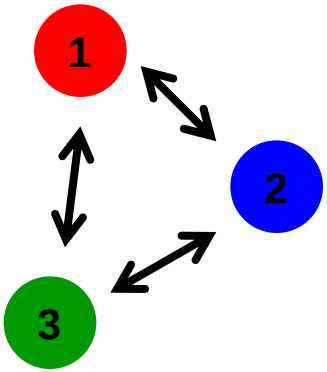
$$\frac{d^2\vartheta(t)}{dt^2} = -\frac{g}{L}\vartheta(t) \rightarrow \frac{d^2\vartheta(t)}{dt^2} = -\frac{g}{L}\sin\vartheta(t)$$



petite oscillation

oscillation à grand angle

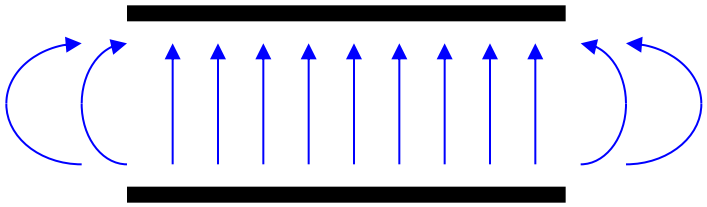
Le problème à trois corps :
Il n'est pas possible de trouver une solution analytique décrivant les trajectoires de trois (ou plusieurs) corps sous l'effet de la gravité.



Il faut résoudre le système d' équations différentielles suivantes:

$$m_i \frac{d\mathbf{v}_i}{dt} = -g_N \sum_{j=1, j\neq i}^{N} \frac{m_i m_j \left(\mathbf{x}_i - \mathbf{x}_j\right)}{r_{ij}^3}$$

Champ électrique d'une capacité à conducteurs plats, mais de dimension finie :
il faut tenir compte aussi des « effets de bord ».



Pour trouver le champ électrique,
il faut résoudre l'équation de Laplace :

$$\nabla^2 \Phi(x, y) = 0$$

avec des conditions aux bords
(équations différentielles à dérivées partielles).

Plan du cours

1. Introduction à C++ : premiers programmes, variables
2. Déroulement d'un programme : conditions et itérations
3. Fonctions
4. Tableaux
5. Intégration et différentiation numérique
6. Pointeurs
7. Structure de l'ordinateur, représentations des nombres
8. Introduction à la résolution des équations différentielles ordinaires
9. Introduction à la programmation orientée objet
10. Résolution numérique des équations différentielles

Organisation du cours

La salle est disponible de 9h00 à 13h00

9h00 – 10h00 correction des exercices de la semaine précédente (facultatif)
(un assistant est toujours présent)

10h00 – 12h00 cours

12h00 – 13h00 séance facultative d'exercices
(un assistant sera toujours présent)

N'hésitez pas à poser des questions !
Il faut bien débuter pour apprendre à programmer.

Si nécessaire, on peut organiser des séances supplémentaires d'exercices.

Méthode d'évaluation

opt. A) 3 contrôles continus consistant en 3 épreuves «écrites» pendant le semestre.

Chaque épreuve consistera en :
des questions
des exercices de programmation
des problèmes de physique à résoudre par ordinateur

Les dates choisies pour les contrôles sont : jeudi 12 mars
jeudi 23 avril
jeudi 28 mai

La note finale est la moyenne des notes des 3 épreuves.

opt. B) Si vous préférez, vous pouvez soutenir un examen à la fin du cours en juin ; l'examen consistera en une éprouve « écrite ».

Pour vous aider, à la fin de chaque leçon, nous vous proposerons des questions, des exercices et des problèmes à résoudre.
Au début de la leçon suivante, nous corrigerons ensemble les exercices.
Si vous avez des questions il faut les poser tout suite et ne pas attendre !

A la fin du cours, vous avez la possibilité d'évaluer le cours par questionnaire.

Textes

textes utilisés
M. Micheloud et M. Rieder
Programmation orientée objets en C++

J.-C. Chappelier et F. Seydoux
C++ par la pratique

textes conseillés
Bjarne Stroustrup
Programmation : Principes et pratique avec C++

référence base
Bjarne Stroustrup
Le langage C++

notes du cours
http://dpnc.unige.ch/~bravar/C++2015

et le web : n'hésitez pas à chercher des réponses à vos questions, des solutions, etc. sur le web. On y trouve aussi plusieurs cours de C++.

(Plusieurs copies des ces textes sont à disposition dans la bibliothèque)

Plan du jour #1

Introduction au cours ; Qu'est-ce que C++ ?

texte Micheloud - Rieder
chap. 1 à 5

Installation du compilateur Dev-C++

Premier programme

Analyse du premier programme

Variables: déclaration
initialisation
affectation
constantes

Les types de données numériques

Opérateurs arithmétiques simples

Entrée / Sortie

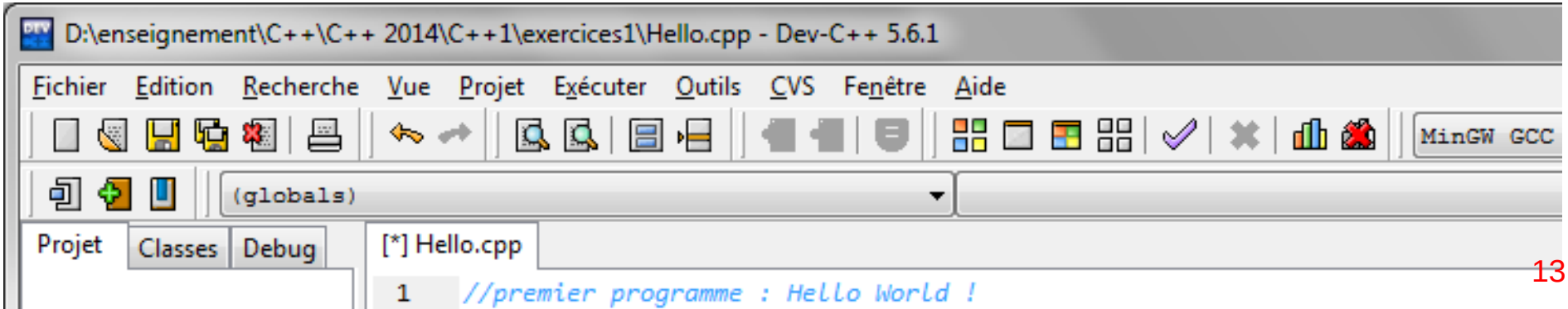
Fonctions mathématiques

Le compilateur C++

Le compilateur est un programme particulier qui “traduit” le programme (fichier source) écrit dans un langage de programmation qui nous est compréhensible (p.ex. C++) en un langage composé de 0 et 1 compréhensible à l’ordinateur.

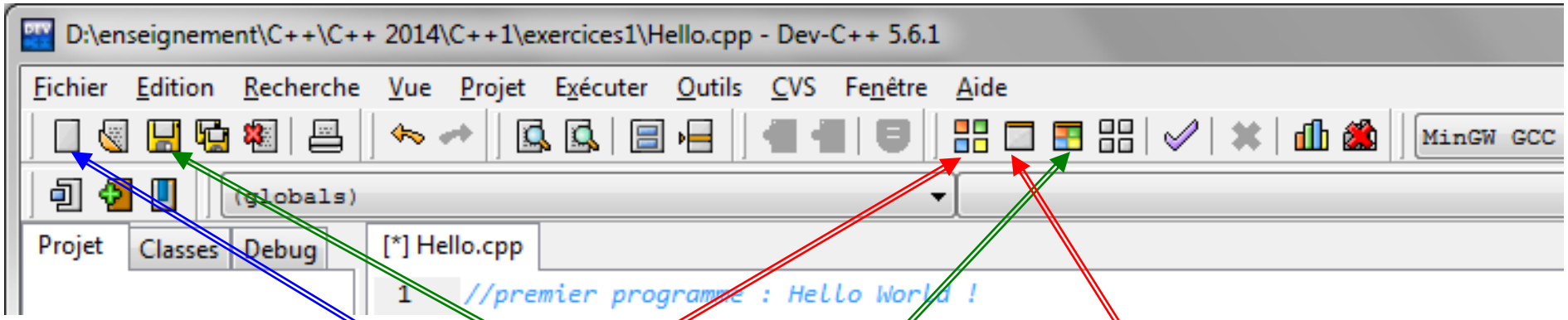
Pour le cours, nous utiliserons la plateforme Windows (plus répandue, mais pas nécessairement la meilleure !). Tous les programmes C++ sont écrits selon la norme ANSI / ISO de 2011 (ANSI C++11). L’objectif du standard ANSI est de garantir la portabilité du code C++ d’une machine à l’autre.
Tous nos programmes fonctionnent sur PC, MAC et Linux.
Le compilateur que nous utilisons est le GCC (GNU Compiler Collection, crée par le projet GNU, open source, donc gratuit). Il est déjà installé sur les ordinateurs de l’Uni.

Pour nous faciliter la tâche, nous utiliserons l’ « interface graphique » Dev-C++.
Dev-C++ est un environnement de développement intégré (IDE) qui fournit aussi un éditeur pour écrire le programme, un debugger pour corriger le programme, etc.
Une fois Dev-C++ lancé une fenêtre comme ci-dessous apparaitra :



L’ interface graphique Dev-C++

L’IDE Dev-C++ est distribué par Orwell (il est gratuit) et il est déjà installé sur les ordinateurs de l’Uni.
Pour lancer Dev-C++, cliquez deux fois sur l’icone
Une fenêtre comme ci-dessous apparaitra



Cliquez sur cette icône pour créer un nouveau fichier source.

Ecrivez le programme comme à la page suivante (Hello.cpp).
Sauvez le programme en cliquant sur l’icône .

Compilez-le (d’abord c’est le compilateur qui est appelé puis l’éditeur de liens).
[un fichier exécutable est créé dans le même dossier que le fichier source]

Si vous n’avez pas commis de fautes, lancez le programme en cliquant sur .

On peut aussi lancer le compilateur et exécuter le programme en même temps en appuyant sur l’icône .

Premier programme

Tout d'abord, il faut écrire le programme (fichier source).
Lancez l'éditeur intégré dans Dev-C++, puis tapez le code suivant :

```
//premier programme : Hello World!
#include <iostream>

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

voir Hello.cpp

Les différentes couleurs montrent les différents composants du programme pour en faciliter la «lecture», p.ex. les mots-clés sont en gras, le commentaires en bleu, les chaînes des caractères en rouge. Il s'agit d'une option de DeV-C++, qui n'a aucun effet sur le programme.

Tout le monde a débuté par ce programme !
Attention ! il faut taper le programme exactement comme ci dessus.

Pour l'instant ne vous posez pas de questions sur la signification de chaque ligne ou instruction : cela viendra au fur et à mesure de vos progrès en C++. Le but aujourd'hui est d'apprendre à créer des programmes, pas encore de tout comprendre … (☺ ou ☹ ?)

Sauvez le fichier sous Hello.cpp (p.ex. dans Mes Documents ou C:\Temp),
puis lancez le compilateur et exécutez le programme.

Si vous n'avez pas commis de fautes, le programme affichera sur l'écran : Hello World!

Analyse

`//premier programme` est un commentaire.
Tout ce qui suit les double barres obliques `//` est ignoré par le compilateur.
Les commentaires peuvent aussi être entourés par `/* ..texte.. */` .
Ils nous aident à mieux comprendre le programme !

La directive `#include` indique au compilateur qu'il doit inclure le fichier `iostream` dans le programme. `iostream` contient la définition de la *fonction* `cout` et fait partie de la bibliothèque standard de C++. Ces fichiers sont fournis avec le compilateur.

`int` `main` est le point d'entrée (ou de départ) du programme. La fonction principale `main` est obligatoire dans tout programme C++. Elle est appelée par le système d'exploitation (e.g. Windows ou Linux). Les accolades { … } contiennent le corps de la fonction principale. A chaque accolade { doit correspondre une accolade } .

`std::cout` est une instruction qui ordonne à l'ordinateur d'afficher la chaîne `"Hello World!"` à l'écran. `"\n"` est le caractère de saut de ligne.

Chaque instruction est toujours terminée avec le `";"`. N'oubliez jamais le point-virgule !

`return` `0` provoque la sortie de la fonction `main` et le renvoi d'une valeur égale à zéro, qui indique au système d'exploitation que le programme a été exécuté normalement.

On peut aussi écrire :

```
//premier programme : Hello World !
#include <iostream>

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

on envoie à l'écran deux chaînes
de caractères au lieu d'une

```
//premier programme
#include <iostream>

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

on utilise deux instructions
pour afficher à l'écran
les deux chaînes sur la même ligne

ou encore :

```
//premier programme : Hello World !
#include <iostream>

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

deux instructions différentes sur la même ligne séparées par le ;

Le résultat est le même ! Essayez !

En général, le compilateur ignore tous les espaces vides (blanc).
On écrit le programme de la façon la plus simple à lire et à comprendre.

* Premier programme

On peut aussi faire sans Dev-C++.

Lancez un éditeur de texte come le notepad (pas Word !), puis tapez le code suivant :

```
#include <iostream>

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

Tout le monde
a initié par ce programme !
Attention ! il faut taper le programme comme ici à gauche.

Sauvez le fichier sous Hello.cpp (p.ex. dans Mes Documents ou C:\Temp)

Maintenant, il faut lancer le compilateur.
Pour faire ça, ouvrez une fenêtre de commande : démarrer → Exécuter → cmd
et compilez le programme avec (-o indique le nom du fichier exécutable)

g++ hello.cpp –o hello ou C:\Programes\Dev-Cpp\MinGW32\bin\g++ Hello.cpp –o hello

Le «résultat» de la compilation est enregistré dans le fichier exécutable hello.
Puis lancez le programme : tapez simplement hello
Si vous n'avez pas commis de fautes, le programme affichera sur l'écran : Hello World!

* Installation du compilateur Dev-C++

Installation de Dev-C++ sur votre ordinateur portable au fixe à la maison :

1a) copiez le compilateur à partir d'une clé mémoire ou
1b) télécharge-le depuis http://dpnc.unige.ch/~bravar/C++2015/compilateur ou
1c) téléchargez-le depuis le lien suivant :
http://orwelldevcpp.blogspot.com/ (il est gratuit)
suivre les liens vers la page de téléchargement et sélectionner la version **DeV-C++ 5.6.1** avec **Mingw32/GCC 4.8.1** (dernière version du DeV-C++ est la 5.9.2)

2) cliquez deux fois sur l'icône d'installation, choisissez la langue et acceptez les conditions de licence (*Accept*)

3) choisissez les composantes Full (complet)

4) choisissez le dossier de destination (C:\Program Files (x86)\Dev-Cpp est recommandé) et *Fermer*

5) lancez Dev-Cpp (cliquez deux fois sur l'icone Dev-Cpp)

6) choisissez la langue, puis le style *Classic* et *New Look*, et «*don't cache anything*» (on modifiera les options du compilateur plus tard (au fur et à mesure de nos besoins)

Si vous avez un MAC ou si vous utilisez Linux, des compilateurs C++ sont déjà installés avec le système. Pour y accéder, tapez simplement gcc ou g++.

* Premier programme (2)

```
//premier programme : Salut
#include <iostream>
#include <ctime>
#include <string>

using namespace std;

int main() {
  cout << "Comment tu t'appeles ?" << endl;
  string prenom;
  cin >> prenom;

  time_t maintenant;
  time(&maintenant);
  cout << "Salut " << prenom << " Aujourd'hui est ";
  cout << ctime(&maintenant) << endl;

  return 0;
}
```

Au lieu du caractère de saut de ligne `\n` nous avons utilisé l'instruction `endl`.

voir Salut.cpp

Cet exemple est plutôt élaboré et avancé par rapport à notre cours.
Pour l'instant ne vous posez pas de questions sur la signification de chaque ligne ou instruction : cela viendra au fur et à mesure de vos progrès en C++. Le but aujourd'hui est d'apprendre à écrire des programmes, pas encore de tout comprendre … (☺ ou ☹ ?)

* Analyse (2)

Dans `iostream`, certaines fonctions sont groupées sous le nom "`std`" (standard). C'est p. ex. le cas de la fonction `cout`. Etant donné que l'on utilise souvent ces fonctions, on place la commande **`using namespace`** `std;` au début du programme. Celle-ci nous permet d'utiliser les fonctions de l'espace de noms "`std`" sans l'opérateur :: (on étudiera plus tard l'espace de noms).

Pour saisir des données depuis le clavier, on utilise la fonction `cin` définie dans `iostream` comme `cout` (utilisé pour la sorite). Les données saisies (ici votre nom) sont mises dans la variable `prenom` de type `string` (objet chaîne de caractères) défini dans le fichier en-tête `<string>` .

Avant d'utiliser n'importe quelle variable, en C++ il faut la définir !
Les caractères et les chaînes des caractères sont toujours entourées par des `" "` .

Pour accéder à la date, nous utilisons la fonction `time` fournie par C++ et définie dans le fichier en-tête `ctime` que nous avons inclus avec la directive `#include <ctime>`. Puis la date est enregistrée dans la variable `maintenant`. Ici, on a défini `maintenant` comme variable de type `time_t` (en réalité, il s'agit d'un objet; nous étudierons les objets et les classes plus tard).
Ensuite nous avons affiché la date avec la fonction `ctime`.

Nous étudierons les fonctions en détail dans quelque leçon.
Pour l'instant, il ne faut pas se faire trop de soucis si tout cela vous semble obscure.

* Compilateur et Editeur de liens

Un programme est une séquence d'instructions (algorithmes) qui agissent sur des données et qui modifient ces données. Le déroulement du programme est contrôlé par ces mêmes données.

Les premiers programmes étaient écrits en code machine (séquence de 0 et de 1 correspondant aux différentes instructions). Très rapidement, on est passé à la programmation en langage assembleur, dans laquelle des expression symboliques remplaçaient les codes numériques.

Des langages évolués (y compris C++) ont été développés pour faciliter la programmation (i.e. surmonter les inconvénients du assembleur). Tous les langages évolués disposent de structures de données et de contrôle permettant aux développeurs de se concentrer sur l'algorithmique des applications. Ces instructions ne sont plus directement compréhensibles par l'ordinateur. Une phase de traduction en langage machine est donc nécessaire pour obtenir un programme exécutable. C'est le rôle du compilateur, un programme qui prend en entrée un fichier texte, contenant le code source écrit dans un langage évolué, pour produire un fichier exécutable compréhensible par la machine (séquence de 0 et 1).

Certains langages (p. ex. le Basic) ne traduisent pas les instructions, mais les exécutent au fur et à mesure via un interpréteur. Pour chaque ligne de code, l'interpréteur simule son exécution. On n'a plus besoin de compiler le programme. Par contre, le programme tourne beaucoup plus lentement que s'il avait été compilé.

Le compilateur génère ce que l'on appelle un fichier objet, qui n'est pas immédiatement exécutable. En effet, le programme est souvent découpé en modules séparés. Une fois compilés, il faut les réunir en une seul application : c'est le rôle de l'éditeur de liens. Ce programme prend tous les fichiers objet, récupère diverses bibliothèques standards et regroupe le tout, en gérant les liaisons entre modules et bibliothèques pour fournir une application complète et exécutable.

Cycle de développement

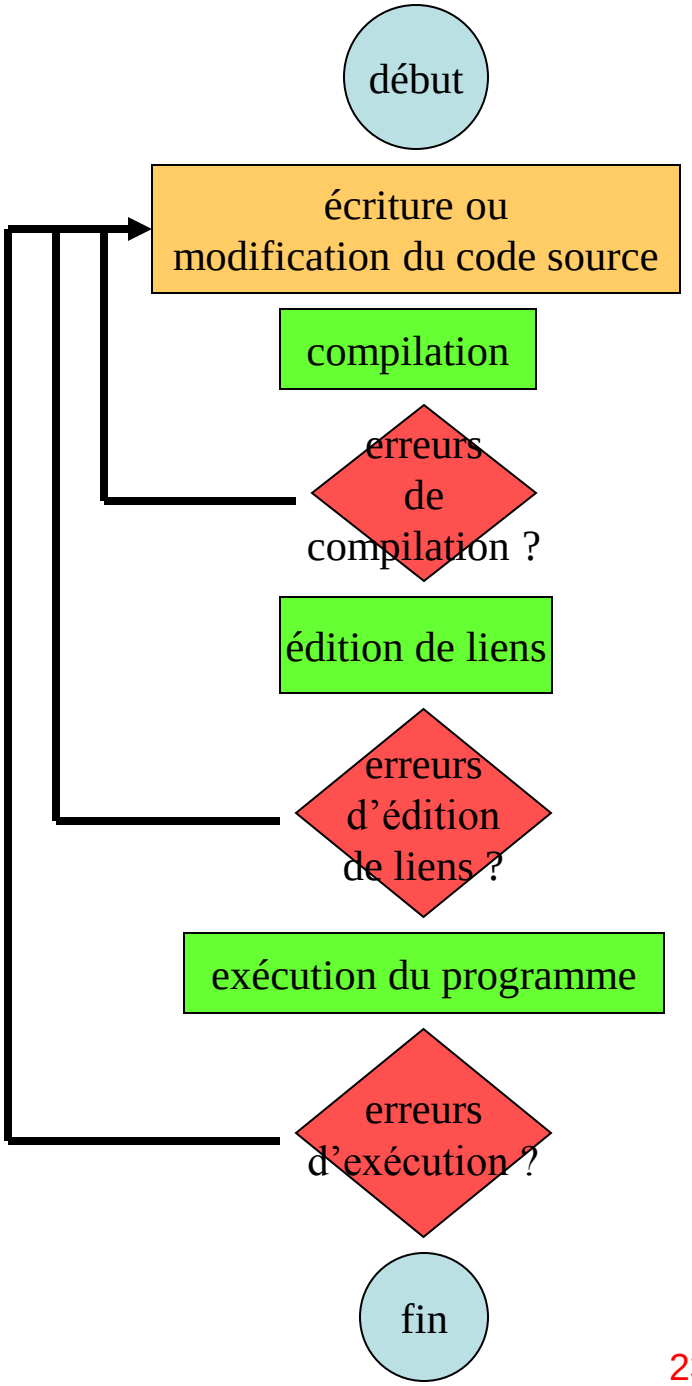
Conception du programme
- modélisation du problème,
en utilisant, p.ex. un pseudo-code
- design du code

Écriture du code source
(éditeur de texte)

Compilation
- préprocesseur
(p.ex. exécution d'instructions #)
- génération d'un fichier objet
(compréhensible par l'ordinateur)

Éditeur de liens
- liaison d'un ou de plusieurs fichiers
avec une ou plusieurs bibliothèques
- création d'un fichier exécutable

S'il n'y a pas d' erreurs → terminé
vous pouvez exécuter le programme ☺



Erreurs (*bugs)*

Si un programme fonctionnait dès sa création, le cycle de développement complet serait mis en œuvre. Toutefois, souvent les programmes contiennent des erreurs ou *bugs*. Les erreurs, plus ou moins graves, empêchent la création du programme ou se manifestent pendant l'exécution. Toute erreur doit être supprimée en modifiant le code source.

Il existe en fait divers types d'erreurs:

Les erreurs de syntaxe

Le programme est mal écrit, p. ex. une accolade ou un point-virgule a été oublié; un mot clé n'a pas été correctement épelé ou une variable n'a pas été déclarée. Ces erreurs sont faciles à trouver, car le compilateur signale le problème et indique la ligne où se trouvent les erreurs.

Les erreurs d'édition de liens

Le compilateur ne trouve pas les fichiers nécessaires à la création du programme exécutable.

Les erreurs d'exécution

La syntaxe est correcte, mais ce que fait le programme est erroné (p. ex. une division par zéro, variable non initialisée correctement, etc.). Ces erreurs sont plus difficiles à trouver, car elles ne sont pas toujours visibles.

Les erreurs de conception

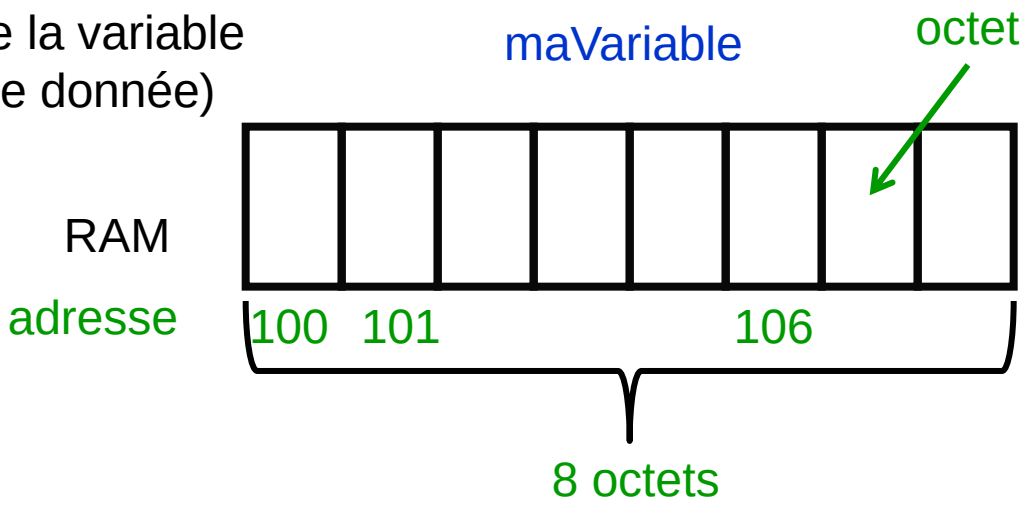
L'algorithme choisi ne fait pas ce que l'on croit.

Toujours il faut lire et essayer de comprendre les messages d'erreur généré par le compilateur, puis les corriger.

Variables

Un programme manipule et modifie des données. Les données sont enregistrées dans des variables. Une variable est un symbole représentant un emplacement de stockage dans la mémoire de l'ordinateur (RAM) destiné à recevoir des données.

Chaque variable occupe de 1 à 8 (voir 16) octets (1 octet = 8 bits) selon le type et est codée selon son type.



Il faut 4 informations pour définir complétement une variable :

- nom
- type (→ taille)
- location
- valeur

le nom, le type et la valeur sont choisis par le programmeur, tandis que la location mémoire de la variable est choisie par l'ordinateur.

Variables

Il existe deux grandes classes de types de données numériques en C et C++ :

1. les types intégraux représentent les nombres entiers ;
2. les types à virgule flottante (floating-point) représentent des approximations des nombres réels.

A chaque type correspond une plage de valeurs.
P.ex. avec 32 bits on peut représenter 2^{32} = 4294967296 valeurs différentes.

Principaux types de variables

entiers avec signe : **`int`** (4 octets)
plage de $-2^{31} = -2147483648$ à $2^{31} - 1 = -2147483647$

entiers sans signe : **`unsigned int`** (4 octets)
plage de 0 à $2^{32} - 1 = 4294967295$

virgule flottante : **`double`** (8 octets)
(mémorisés sous forme exponentielle : mantisse et exposant)
plage de 2.22507^{-308} à 1.79769^{+308} (avec signe)

caractères : **`char`** (1 octet) codé selon le jeu de caractères ASCII (128 symboles)

booléen : **`bool`** (1 octet) : 2 valeurs possibles : `true` (vrai) ou `false` (faux)

Déclaration et Affectation

Avant d'utiliser une variable en C++, il faut toujours la déclarer !

déclaration d'une ou de plusieurs variables

```
type  nom_de_la_variable;
```

```
int i;
int j;
int k;
```
ou
```
int i, j, k;
```
(même type, une seule instruction)

```
int i; double x;
```
(2 types différentes → 2 instructions différentes !)

affectation des valeurs (et initialisation)

l'affectation est l'opération par laquelle on attribut une valeur à une variable

```
double x;
x = 5.9;
```
ou
```
double x = 5.9;
```
(déclaration et initialisation)

```
double x = 3.1, y = 4.5;
double z;
z = x * y;
```
ou
```
double x = 3.1, y = 4.5, z = x * y;
```

déclaration d'une constante (`const`)

```
const double PI = 3.14159;
```
la valeur de `PI` ne peut plus être changée

« arithmétique » de l'ordinateur

Un programme manipule et modifie des données (il ne fait que ça !).
On travaille toujours avec des expressions comme :

i) `x = 5.;`

ou

ii) `x = x + 5.;` (la nouvelle valeur enregistrée dans `x` est 10)

ou encore

iii) `x = y = x + 3.;` (la valeur enregistrée dans y et dans x est 13)

Evidemment, il ne s'agit pas d'équations algébriques, mais d'affectations.
= est l'opérateur d'affectation et non plus l'opérateur égalité de l'arithmétique.
L'affectation procède toujours de droite à gauche !

P. ex. dans ii) on prend la valeur enregistrée dans la variable `x` (qui avant a été déclarée de type double !), on y additionne 5, puis on enregistre le résultat dans la même location de mémoire de la variable `x`. L'ancienne valeur de `x` est perdue.

Aussi l'instruction iii) est admissible (mieux utiliser deux instructions !) : on prend la valeur enregistrée dans la variable `x`, on y additionne 3, puis on enregistre le résultat dans la variable `y`, puis on enregistre la même valeur dans la variable `x`.

```
int n = 5;
n = n + 2;      n = ?
n = 3 * n;      n = ?
```

il ne s'agit pas d'une équation mathématique !

Comment bien choisir les noms des variables

Attention ! C++ fait la différence entre les majuscules et les minuscules (case sensitive).

p. ex. `variable` ou `Variable` ou `VARIABLE` ou `variaBle` sont des noms de variables différentes !

Un nom de variable peut être composé de n'importe quelle combinaison de lettres (non accentuées) de chiffres et le tiret bas _ . Le premier caractère doit être une lettre.

p. ex. `x  xzy123  Test  monAge  deg_kelvin  JWT703  _var`

sont tous des noms admissibles. Par contre

`1  4xy  x?y  deg Kelvin  carré`

ne sont pas admissibles comme noms de variables.

Il faut ainsi essayer d'utiliser des noms suffisamment explicites (mais pas très longs) qui reflètent le rôle plutôt que l' implémentation.

Evitez les noms *bizarres* qui n'évoquent rien comme `JWT703`. Evitez aussi des lettres isolées comme `x` ou `i`, sauf si leur utilisation est évidente. Evitez aussi le noms commençant par _ . En revanche, des noms explicites comme `degreKelvin` ou `monAge` sont faciles à comprendre. Les constantes sont d'habitude écrites en majuscules (p. ex. `PI`), les classes débutent par C…, etc.

Mais le plus important est de toujours suivre la même convention dans le programme.

Opérateurs arithmétiques simples

Un operateur est un symbole qui opère sur une ou plusieurs expressions et produit une valeur. On utilise les même opérateurs pour les entiers et le flots (sauf le modulo).

Opérateur	Description	Exemple (`int m=38, n=5;`)	Résultat
+	addition	m + n	43
–	soustraction	m – n	33
–	négation	–m	–38
*	multiplication	m * n	190
/	division	m / n	7 !
%	reste (modulo)	m % n	3

Opération entre entiers.
Le résultat est aussi un entier !

Opérateurs d'affectation composés

au lieu d'écrire

```
n = n + 2;      et      m = 3 * m;
```

on peut aussi écrire (plus pratique !)

```
n += 2;         et      m *= 3;
```

L'ordre de calcul des expressions suit les règles de l'arithmétique et peut être modifié avec des parenthèses ( ) : p. ex. (3 + 5) * 9 .

Exemple d'affectation et d'arithmétique

```
//exemple d'affectation
#include <iostream>
using namespace std;

int main() {
  //declaration et affectation avec deux instructions
  int i;
  i = 10;

  //declaration et initialisation au meme temps
  double x = 20.;

  //delcaration d'une constante
  const double PI = 3.14159;

  //et avec une operation arithmetique
  int k = i * (i + 2) + 5;      //k = 125
  double y = x * x;      //y = 400.

  //affichage (sortie sur l'ecran)
  cout << "Carre de " << x << " : " << y << endl;
  cout << "Cube de " << i << " : " << i * i * i << endl;
  cout << "k fois constante : " << k * PI << endl;
  return 0;
}
```

En général, il est toujours préférable de initialiser les variables lors de la déclaration.

déclaration de la variable entier `i`
affectation de la valeur 10 à `i`

déclaration et initialisation de la variable double `x`

déclaration d'une constante double

Les constantes doivent être initialisées lors de la déclaration !
Elles ne peuvent plus être modifiées.

voir Affectation.cpp

Opérations entre entiers et virgules flottantes

Une opération arithmétique entre entier donne comme résultat un entier :

```
int m = 38, n = 5;    cout << m / n;
```
affichera 7 !

Une opération entre des nombres à virgule flottante, donne comme résultat un nombre à virgule flottante (pour indiquer qu'il s'agit d'un *float* on ajoute le point décimal) :

```
double m = 38., n = 5.;    cout << m / n;
```
affichera 7.6 !

Dans une opération entre entiers et virgule flottantes, les entiers sont d'abord convertis en flots (parfois avec perte de précision). On dit que les entiers sont ainsi promus vers un type « supérieur ». Le calcul qui suit est fait entre flots.

`3/4.5` → `3.0/4.5 = 0.666` `(3/4 = 0 !)`

mais `3.0/4.5 + 9/7 = 0.666 + 1` → `1.666` (et non pas `1.952` !)

La conversion d'un type vers un autre s'appelle transtypage (casting). La conversion vers un type supérieur ne pose pas de problèmes, sauf une perte de précision si le nombre est très grand. Par contre, la conversion d'un flot vers un entier implique une troncature et pas un arrondi :

`3.5` → `3`, `3.1` → `3`, `3.9` → `3`, seulement la partie entière est retenue.

En général, si `T` est d'un type particulier et que `v` représente une valeur d'un autre type, l'expression `T(v)` convertit `v` en type `T`, p. ex. :

```
double x;
int n = int(x);  ou  int n = x;
```
convertit la valeur de `x` en entier.

Exemple d'opérations entre types différents

```
//exemples de transtypage
#include <iostream>

using namespace std;

int main() {
  cout << "a = 2/3 :" << 2 / 3 << endl;
  cout << "b = 2/3.0 : " << 2 / 3.0 << endl;
  cout << "c = 2.0/3.0 : " << 2.0 / 3.0 << endl;
  cout << "d = int(2.0)/3.0 : " << int(2.0) / 3.0 << endl;
  cout << "e = int(2.0)/int(3.0) : " << int(2.0) / int(3.0) << endl;
  cout << "f = 2.0/int(3.0) : " << 2.0 / int(3.0) << endl;
  cout << "g = double(2)/double(3) : " << double(2) / double(3) << endl;

  int n = 987654321;
  double x;
  x = n;
  int k;
  k = x;
  cout << "n : " << n << endl;
  cout << "x : " << x << endl;
  cout << "k : " << k << endl;

  return 0; }
```

pendant l'affectation de la variable double `x`,
la valeur de la variable entier `n` est convertie en flot

la valeur de `double x` est convertie en entier
avec perte de précision

on ne retrouve pas les mêmes valeurs !

voir Cast.cpp et Cast2.cpp

* Types de données numériques

Il existe deux grandes classes de types de données numériques en C et C++ :
1. les types intégraux représentent tous les nombres entiers ;
2. les types à virgule flottante (floating-point) représentent des approximations des nombres réels.

A chaque type correspond une plage de valeurs.

Types intégraux:

Les types intégraux sont les suivants:

- bool 8 bits
- char 8 bits
- short 16 bits
- int 32 bits
- long 32 bits
- long long 64 bits

et existent aussi dans la version avec ou sans signe (signed ou unsigned).

Types à virgule flottante:

Les nombres à virgule flottante sont les nombres le plus souvent utilisés pour représenter des valeurs non entières. Ce sont des approximations de nombres réels.

Il en existe de trois types:
- float (valeur à 32 bits, précision dite simple)
- double (valeur à 64 bits, précision dite double)
- long double (valeur à 80 bits)

* Taille des variables

L' operateur `sizeof` renvoie la longueur en octet d'une variable.
Les intervalles de définition sont définis dans les fichiers en-tête `cfloat` et `climits`.

```
.#include <iostream>
 #include <cfloat>      //contient les limites
 #include <climits>      //contient les limites
 using namespace std;

 int main() {
   cout << "The size of an int is: \t\t\t";
   cout << sizeof(int) << " bytes.\n";
   cout << "The size of a short int is: \t\t";
   cout << sizeof(short) << " bytes.\n";
   cout << "The size of a long double is: \t\t";
   cout << sizeof(long double) << " bytes. \n\n";

   cout << "The min / max values of an int are: \t";
   cout << INT_MIN << "\t" << INT_MAX << "\n";
   cout << "The min / max values of a float are: \t";
   cout << FLT_MIN << "\t" << FLT_MAX << "\n";
   cout << "The min / max values of a double are: \t";
   cout << DBL_MIN << "\t" << DBL_MAX << "\n\n";

   return 0;
 }
```

`\t` et `\n` sont des caractères de contrôle utilisés dans la mise en page

voir Taille.cpp

* Les caractères

Les variables caractères (`char`) sont codifiées selon le jeu de caractères ASCII (voir p. ex. http://fr.wikipedia.org/wiki/American_Standard_Code_for_Information_Interchange) et sont enregistrées sur un octet (128 valeurs possible). Le jeu de caractères ASCII permet de coder toutes les lettres (minuscules et majuscules), les chiffres, les signes des ponctuation et les caractères de contrôle.
Le standard UNICODE (wchar_t) désigne un jeu de caractères plus étendu sur deux octets (~65536 valeurs possible), qui permet de codifier tous les alphabets du monde, plusieurs symboles, etc.

Pour affecter un caractère à une variable de type caractère (`char`) on utilise le caractère en question entre apostrophes ou son code ASCII :

```
char c = 'S';
```
ou
```
char c = 83;
```

Dans tout les deux cas, l'instruction suivante affichera la lettre S.

```
cout << c;
```

Et pour lire un caractère :

```
char z;
cin >> z;
```

L'on peut aussi faire de l'arithmétique avec les caractères. P. ex.

```
cout << c + 1;
```

affichera la lettre T (mais il ne faut pas exagérer ☺).

* Davantage sur les types de données

types synonymes

Un type synonyme est un alias d'un type existant ; il n'introduit pas un nouveau type :

```
typedef indentificateurType nouvelIdentificateur;
```

p.ex.:

```
typedef int entier;
```

Au lieu d'ecrir

```
int n = 10;
```

dans la déclaration des variables, nous pouvons remplacer `int` par `entier` :

```
entier n = 10;
```

types d'énumération

Le type énumération est un type intégral qui permet de définir un type de donnée dont les valeurs possibles sont constituées d'une liste de constantes désignées par des identificateurs :

```
enum identificateurType {liste des énumérateurs};
```

Chaque énumérateur représente une constante entière nommée. La valeur de chaque constante énumérateur est par défaut son numéro d'ordre dans la liste (0, 1, 2, …)

p.ex.:

```
enum mois {janvier, fevrier, mars, avril};
```

L'énumérateur `janvier` a la valeur entière 0, `fevrier` la valeur 1, etc.

Cette instruction équivaut à écrire :

```
const int janvier = 0;   const int mars = 2;  …
```

Entrée/Sortie

Pour être utile, un programme doit pouvoir communiquer avec son environnement.

Pour lire des données depuis le clavier on utilise la fonction `cin`

et pour afficher des données sur l'écran on utilise la fonction `cout`.

Ces deux fonctions sont définies dans le fichier en tête `iostream`.

lecture des données :

```
cin >> argument1 >> argument2 >> argument3;
```

`>>` est l'opérateur d'éntrée

`argument1, argument2, ...` sont des variables définies avant l'utilisation de `cin`

Chaque variable doit être saisie indépendamment des autres, i.e. avec l'operateur `>>`.

affichage des données :

```
cout << argument1 << argument2 << argument3;
```

`<<` est l'opérateur de sortie

`argument1, argument2, ...` sont des variables
des expressions, p.ex. `x+y` ou `3*x`, ...
des chaînes (entre " et "), p.ex. `"salut"`

Pour information, on peut aussi utiliser les fonctions `scanf` et `printf` du langage C définies dans le fichier en tête `cstdio`.

Exemple d'Entrée/Sortie

```
//conversion de degres Centigrade en degres Fahrenheit
#include <iostream>

using namespace std;

int main() {
  //saisi de la temperature
  double temp;
  cout << "Quelle est la température en C ?\n";
  cin >> temp;

  //conversion
  double faren;
  faren = (temp * 9./5.) + 32.;
  cout << "La température en F est : " << faren << endl;

  return 0;
}
```

sortie

entrée

on a écrit `9./5.` au lieu de `9/5`

Pourquoi ?

voir Fahrenheit.cpp

On peut faire la conversion en même temps que la déclaration de la variable `faren`

```
double faren = (temp * 9./5.) + 32.;
```

ou encore au moment de l'affichage (on n'a pas besoin de la variable `faren`)

```
cout << (temp * 9./5.) + 32.;
```

Le résultat est toujours le même. Essayez !

Les fonctions mathématiques

Pour calculer p. ex. `sin(x)` on utilise les fonctions mathématiques fournies par C++.

La bibliothèque standard C++ met à disposition les fonctions mathématiques les plus utilisez. Elle sont définies dans le fichier `cmath`.

Pour les utiliser, il faut ajouter `#include <cmath>` en tête du programme.

abs	ceil	floor	log	sin
acos	cos	fmod	log10	sinh
asin	cosh	frexp	modf	sqrt
atan	exp	labs	pow	tan
atan2	fabs	ldexp	pow10	tanh

P. ex. : `y = sin(x)` renvoie le sinus de `x`, `x` étant donné en radians

`y = pow(x,3.5)` $= x^{3.5}$

Plusieurs constantes mathématiques y sont aussi définies :

`M_PI` = π = 3.14159 …
`M_E` = e = 2.71828 …
`M_LN10` = ln(10) = 2.30258 …

p. ex. pour calculer la surface d'un cercle :

```
M_PI * pow(r,2.);
```

Exemple utilisant une fonction mathématique

```
#include <iostream>
#include <cmath>      //bibliotheque mathématique

using namespace std;

int main() {
  //saisi des donnees
  double rayon;
  cout << "Quel est le rayon de la sphère ? ";
  cin >> rayon;

  //calcul de la surface de la sphere
  const double PI = 3.14159;
  double surface;
  surface = 4. * PI * rayon * rayon;
  cout << "La surface de la sphere est: " << surface << endl;

  //calcul du volume de la sphere
  double volume = 4./3. * M_PI * pow(rayon,3.);
  cout << "Le volume de la sphere est: " << volume << endl;

  return 0;
}
```

déf. de la constante π
sa valeur ne peut plus être modifiée

fonction mathématique `pow`
+ constante M_PI (= π)

voir Sphere.cpp

Style de programmation

Quand vous écrivez un programme, il faut imaginer que quelqu'un d'autre puisse le lire. Il faut donc écrire de manière compréhensible, faire attention à la mise en page, bien choisir le nom des variables et des fonctions, ajouter des commentaires si nécessaire …

Commentaires

Ils sont essentiels pour comprendre le code du programme. Ils sont ignorés par le compilateur et servent à ajouter des explications à l'attention de l'utilisateur (vous-même dans quelques mois !). Ceux-ci peuvent être introduits de deux façons :

1. tout le texte qui suit une double barre oblique // jusqu'à la fin de la ligne est un commentaire,
2. en entourant le commentaire par les symboles /* et */ sur une ou plusieurs lignes sans imbrication (le commentaire débute par /* et est terminé par */); hérité du langage C.

N' hésitez pas à utiliser des commentaires dans votre programme. Dans quelques mois, vous pourriez avoir oublié ce que vos programmes font.

Nom des variables

Attention ! C++ fait la différence entre les majuscules et les minuscules (case sensitive).

Un nom de variable peut être composé de n'importe quelle combinaison de lettres et de chiffres, à l'exception d'espaces. Il faut ainsi essayer d'utiliser des noms suffisamment explicatifs (mais pas très longs) qui reflètent leurs rôle plutôt que leurs implémentation. Evitez les noms *bizarres* qui n'évoquent rien comme `S37ergz`, ou des lettres isolées comme `x` ou `n`. En revanche, les noms explicites comme `total` ou `monAge` sont faciles à comprendre. Les constantes sont d'habitude écrites en majuscules, les classes débutent par C…, etc. Mais le plus important est de toujours suivre la même convention (style de programmation). Même pour les fonctions ou les classes.

Mots-clés du langage C++

Les mots-clés sont interprétés par le compilateur comme des éléments intrinsèques au langage de programmation. Les mots-clés ne peuvent donc pas être utilisés comme noms de variables, de fonctions ou de classes.

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

en rouge, les mots-clés vus aujourd'hui

à retenir : const, double, long, return

Résumé

Ce qu'il faut retenir / savoir faire à la fin de cette leçon :

utiliser un compilateur C++, p.ex. Dev-C++ sur Windows

écrire correctement des programmes

utiliser les variables **`int`** et **`double`**
et comprendre la différence entre entier et flots (virgule flottante)

comprendre la différence entre une variable et une variable constante

faire des calculs simples en utilisant
1. les opérateurs arithmétiques
2. les fonctions mathématiques

«communiquer» avec le programme
entrée depuis le clavier : fonction `cin`
sortie sur l'écran : fonction `cout`

Mémorisez le programme suivant !

Il contient tous les élément importants vus aujourd'hui.

```
//calcul de la racine carree
#include <iostream>
#include <cmath>

using namespace std;

int main() {
  double x, y;
  cout << "Entrez deux nombres > 0 :" << endl;
  cin >> x >> y;

  double w;
  w = x + y;  //on peut aussi ecrire : double w = x + y;
  double z;
  z = sqrt(w);  //on peut simplement ecrire : double z = sqrt(x+y);

  cout << "La racine carree de " << x+y << " est " << z << endl;
  /*on peut aussi ecrire :
  cout << "La racine carree de " << x+y << " est " << sqrt(x+y) << "\n";
  */

  return 0;
}
```

voir Racine.cpp

Exercices – série 1

Questions

1. Quel est le plus petit programme C++ que vous pouvez écrire ? Que fait-il ?

2. Quelles sont les manières possible pour soustraire 1 de la valeur entière n ?

3. A quoi sert l'éditeur de liens ? Quelle est la différence entre compilateur et interpréteur ?

4. Quels sont les deux types de commentaires admis en C++ ?

5. Quelle est la différence entre une variable entière et une variable à virgule flottante ?

6. Ecrivez l'instruction qui affiche le caractère ASCII correspondant à 100.

7. Initialisez un caractère (`char`) à la valeur 65 de deux manières différentes.

Trouvez l'erreur !

Trouvez les erreurs dans ces programmes (écrivez ces programmes et corrigez-les) :

1.
```
#include <iostream>
using namespace std;
int main() {
  Affiche "Hello World"
  cout << "Hello World !\n";
  return 0
```

2.
```
#include <iostream>
int main() {
   int n = 22;
   cout << n << endl;
   RETURN 0;
}
```

3.
```
#include <iostream>
int main()
{
   int n = 22
   cout << n << endl;
};
```

4.
```
#include <iostream>
Using namespace std;
int main() {
   int m = 3;
   n = 22;
   cout << m+n << endl;
   Return 5;}
```

Exercices

1. Ecrivez et exécutez le programme Hello.cpp de la page 15 ou 18.
Essayez les alternatives proposées (page 17).

2. Ecrivez le programme Salut.cpp de la page 21.

3. Etudiez / écrivez le programme Affectation.cpp de la page 31.

4. Ecrivez un programme qui affiche la somme, la différence, le produit, la division et le reste de deux entiers (type `int`). Initialisez les entiers avec les valeurs 60 et 7.
   Ecrivez le même programme avec des variables de type `double` (sauf le reste).

5. Etudiez / écrivez le programme Fahrenheit.cpp de la page 39.

6. Ecrivez un programme similaire pour convertir les angles de degrés en radians.

7. Ecrivez un programme pour convertir les livres en kg (1 lb = 0.453 kg).

8. Mettez des erreurs dans le programme Fahrenheit.cpp de la page 39 :
p.ex. 1) oubliez un ; à la fin de quelque instruction, 2) oubliez une accolade }, …
   Que se passe-t-il ?
   Si le facteur de conversion 9. / 5. est faux, le programme fonctionnera, mais donnera un faux résultat (erreur de conception) !

9. Etudiez le programme Sphere.cpp de la page 41. Ecrivez un programme similaire pour calculer la surface et le volume d'une cube.

10. Ecrivez votre version du sin(x) et cos(x) avec une expansion de Taylor au 4ème ordre. Comparez les résultats obtenus avec les fonctions de la bibliothèque C++.

Problèmes

1. Cinématique :

Ecrivez un programme qui calcule la jetée d'un projectile en fonction de sa vitesse initiale : le programme saisit la vitesse *v* et l'angle θ de jetée par le clavier puis retourne la jetée et le temps correspondant à cette distance ainsi que la hauteur maximale atteinte par le projectile. Le programme saisit ensuite un temps compatible avec le point précédent et retourne la position du projectile en ce temps.

2. Optique :

Un rayon laser est réfléchi par une sphère come montré dans la figure.
Ecrivez un programme pour calculer l'angle de réflexion α. Ou début, le programme saisit Le rayon R de la sphère et le paramètre d'impact b du rayon laser.