Série 2 :
Première programmation

Buts

Cette série a pour but de vous familiariser avec les divers outils dont vous aurez besoin pour écrire et compiler vos programmes en C++, en vous faisant pratiquer sur des exemples simples de (premiers) programmes.

Informations

Pour vous aider dans votre pratique (voire votre [auto-]évaluation), un thème et un niveau de difficulté (de 1 = facile à 3 = avancé) est associé à chaque exercice. Pratiquez donc en priorité les thèmes avec lesquels vous vous sentez le moins à l'aise. De même, si vous êtes doué pour la matière, attaquez directement les exercices niveau 2 et 3.
Si par contre vous êtes en difficulté, focalisez vous sur les niveau 1 et 2 puis revenez plus tard sur le niveau 3.
Notez que les niveaux sont déterminés en fonction du moment où la série est prévue... il est clair qu'un même exercice donné plus tard au cours de l'année (par exemple au moment de l'examen) serait considéré comme beaucoup plus facile !
Il existe de plus des exercices de niveau 0, qui reprennent pas à pas un exemple du cours et l'expliquent en détail. Ces exercices sont prévus pour vous faire assimiler le cours à votre rythme, en particulier aux débutants. Commencez donc par là si vous êtes en difficulté.

La totalité de la série n'est pas prévue pour être finie en 2 heures ! La série est à considérer comme un regroupement de plusieurs exercices, un peu comme un chapitre de livre d'exercices, où chacun peut choisir (à l'aide des indications ci-dessus) ce qui lui semble adapté à sa progression.

Le niveau **moyen** pour 2 heures d'exercices correspond à deux exercices niveau 1 et un exercice niveau 2.

Pour finir, je vous conseille également, en plus des 2 heures d'exercices hebdomadaires à l'emploi du temps, de travailler par vous-même chez vous de façon régulière en guise d'exercices personnels. Ce travail supplémentaire devrait en moyenne correspondre à 2 à 3 heures hebdomadaires.

Pour résumer :

Niveau 0
: Reprise pas à pas d'un exemple du cours. Les exercices de ce niveau peuvent sans problème être sautés par tous ceux qui estiment avoir une suffisamment bonne compréhension de la programmation de base et du cours.

Niveau 1
: Ces exercices élémentaires devraient pouvoir être faits par tous dans un temps raisonnable (30 à 45 minutes maximum). Ils permettent de travailler les bases.

Niveau 2
: Ces exercices plus avancés devraient être abordés par tous, sans forcément être finis. La reprise de l'exercice avec la correction devrait constituer un bon moyen de progresser.

Niveau 3
: Ces exercices d'un niveau avancé sont pour les plus motivés/habiles d'entre vous. Ils peuvent **dans un premier temps** être ignorés. Ils devraient être repris, si nécessaire avec la correction, lors des révisions.

Préliminaires :

Avant d'effectuer les manipulations décrites dans cette série, créez le répertoire `~/Desktop/posixfs/cpp/serie02` (c.-à-d. créez le sous-répertoire `serie02` dans le répertoire `~/Desktop/posixfs/cpp`).

Cette série est volontairement courte (ce ne sera plus le cas dans le futur) pour vous permettre de bien prendre en main votre environnement :

- revoyez les points laissés de coté ou difficiles de la série de la semaine passée ;
- naviguez dans toute la documentation (cf **exercice 10 semaine passée**) ;
- faites les **quiz du MOOC**

Préambule : configuration du compilateur et de l'environnement de compilation sous myfiles

1. Norme du compilateur

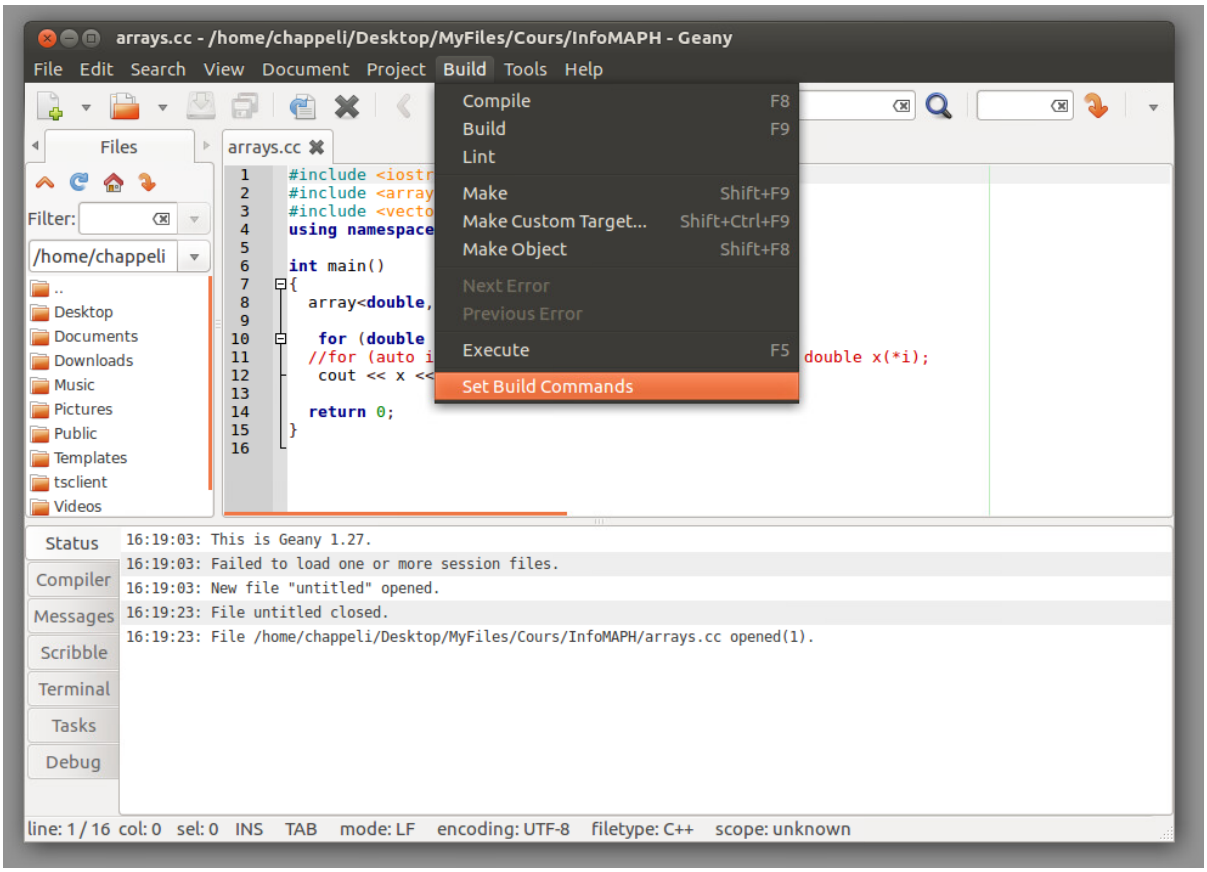
Pour tout le monde (quelque soit votre façon de travailler) :

Ce cours utilise la norme dite « C++ 11 » du C++. Ce n'est pas encore très important pour le moment, mais si vous voulez éviter des erreurs de compilation plus tard, nous vous invitons à **configurer votre compilateur** pour qu'il compile suivant cette norme ou une norme ultérieure.

Cela se fait en ajoutant simplement l'option `-std=c++11` au compilateur (vous pouvez aussi très bien choisir `-std=c++14` ou `-std=c++17`). **Note :** je ne parle pas ici de ceux qui subissent le bug MinGW sous Windows et doivent garder `-std=gnu++11`.

Par exemple dans Geany :

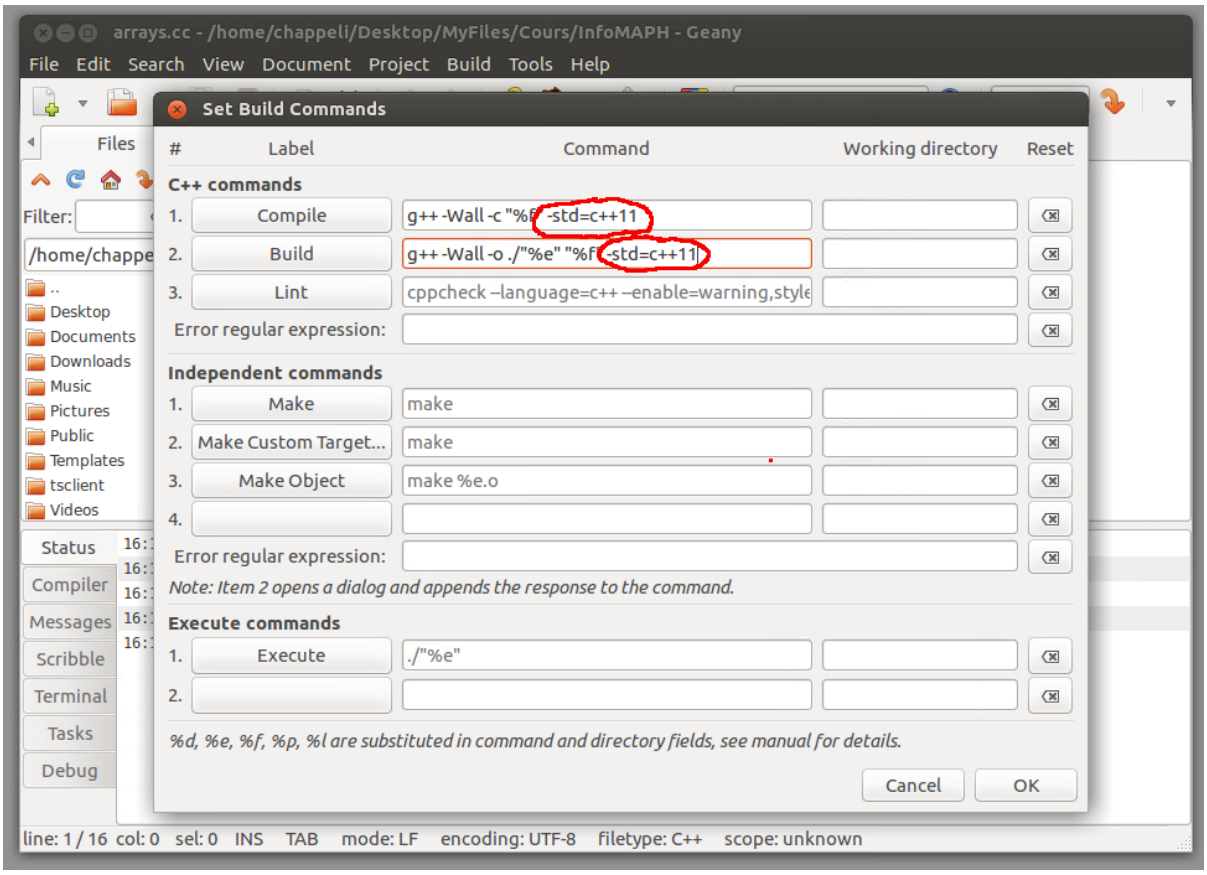
- aller dans le menu : *Construire -> Définir les commandes de construction* (*Build -> Set Build Commands*)



- Dans la colonne *Commande* (*Command*), rajouter `-std=c++11` en fin de ligne dans les deux lignes *Compiler* (*Compile*) et *Construire* (*Build*) :

  ```
  g++ -Wall ... -std=c++11
  ```

  (**attention** à bien mettre **une espace avant le `-std`** !)



- Valider ensuite pour enregistrer les modifications.

2. Spécificité des VMs (salles CO)

Pour ceux travaillant sur les VMs fournies par l'Ecole :

pour pouvoir compiler (en fait : exécuter le résultat de votre compilation) sur ces machines, il est impératif de travailler sous un répertoire ayant des droits particuliers. Pour cela, le répertoire `posixfs` a été crée et vous devez travailler dans ce répertoire comme expliqué à **la fin de l'exercice 5** de la semaine passée.

3. Plus d'outils sur Geany

Pour ceux qui utilisent Geany :

L'éditeur Geany est très configurable. Il peut être bon pour votre productivité d'ajouter plusieurs modules intéressants. Allez dans *Tools* -> *Plugin Manager* et choisissez ceux qui vous parlent.

Personnellement, je vous conseille :

- Addons
- Auto-close
- Debugger
- File Browser

Exercice niveau 0

> Une variante de cet exercice est disponible en page 8 de l'ouvrage *C++ par la pratique*

Le but de cet exercice est de vous apprendre, pas à pas, à

- éditer un programme ;
- compiler un programme ;
- corriger les erreurs de compilation.

Cliquez ici si vous voulez faire cet exercice.

Second exercice niveau 0 (IMC)

Pour cette toute première série de programmation, je vous propose un second tutoriel («exercice niveau 0») si nécessaire : allez voir l'exercice de l'IMC dans le tutoriel de **la semaine 1 du MOOC**.

Exercice 1 : Variables (niveau 1)

> Exercice n°1 (pages 13 et 195) de l'ouvrage *C++ par la pratique*
> (pages 13 et 197 dans la 2e édition).

> Exercice n°1 du MOOC

Écrivez un programme `age.cc` qui :

1. demande son âge à l'utilisateur ;
2. lit la réponse de l'utilisateur et l'enregistre dans une variable *age* de type entier ;
3. calcule l'année de naissance (à un an près) de l'utilisateur et l'enregistre dans la variable *annee* de type entier ;
4. affiche l'année de naissance ainsi calculée.

Compilez et exécutez votre programme (voir **exercice précédent** pour les détails, mais en résumeé : `F9` dans Geany, «Compile» dans Emacs, puis `F5` dans Geany ou `./age` dans le terminal).

Indications :

- Pour demander à l'utilisateur son âge, affichez un message à l'aide de l'instruction suivante :

  ```
  cout << "Quel age avez-vous ? " ;
  ```

- Pour lire la réponse de l'utilisateur et l'enregistrer dans la variable *age*, utilisez l'instruction suivante :

  ```
  cin >> age;
  ```

- Pour afficher du texte et des variables, séparez chaque élément par `<<`. Exemple :

  ```
  cout << "message 1 " << var1 << ", message2 " << var2 << endl;
  ```

  (le `endl` à la fin correspond au «saut de ligne» et permet à l'affichage de passer à la ligne suivante).

Exercice 2 : Fondue (niveaux 1)

> Exercice n°2 du MOOC

Le but de cet exercice est d'écrire un programme qui permet d'adapter automatiquement, en fonction du nombre de convives, les quantités d'ingrédients nécessaires à la confection d'une fondue fribourgeoise (au pur Vacherin fribourgeois ; une recette typiquement suisse, mais à ne pas confondre avec la fondue « moitié-moitié », suisse également `;-)`).

Ecrivez un programme `fondue.cc` qui :

1. déclare une constante `BASE`, initialisée à 4, et qui indique le nombre de personnes pour laquelle est conçue la recette de base ;
2. déclare une variable `fromage`, initialisée à 800.0, qui donne la quantité de fromage en grammes nécessaire pour `BASE` personnes (du « Vacherin fribourgeois » en l'occurrence !) ;
3. déclare une variable `eau`, initialisée à 2.0, qui donne la quantité d'eau en décilitres nécessaire pour `BASE` personnes ;
4. déclare une variable `ail`, initialisée à 2.0, qui donne le nombre de gousses d'ail nécessaires pour `BASE` personnes (on choisit le type `double` car on veut pouvoir utiliser des moitiés de gousses par exemple) ;
5. déclare une variable `pain`, initialisée à 400.0, qui donne la quantité de pain en grammes nécessaire pour `BASE` personnes ;

6. demande à l'utilisateur d'introduire le nombre de convives pour lequel on veut préparer la recette ;
7. lit la réponse de l'utilisateur et l'enregistre dans une variable `nb_convives` de type entier ;
8. adapte les quantités de chaque ingrédient en faisant une règle de trois (`nouvelle_quantite = quantite_de_base * nb_convives / BASE`);
9. et affiche la recette pour le nombre de convives voulus selon l'exemple ci-dessous.

Exemple d'exécution du programme

```
Entrez le nombre de personne(s) conviées à la fondue : 3

Pour faire une fondue fribourgeoise pour 3 personnes, il vous faut :
 - 600.0 gr de Vacherin fribourgeois
 - 1.5 dl d'eau
 - 1.5 gousse(s) d'ail
 - 300.0 gr de pain
 - du poivre à volonté
```

Exercice 3 : Variables (niveaux 2 (début) et 3 (explication))

> Exercice n°2 (pages 13 et 196) de l'ouvrage *C++ par la pratique*
> (pages 13 et 198 dans la 2e édition).

> Exercice n°3 du MOOC

Écrivez un programme `calcul.cc` qui :

1. déclare les variables *x* et *y* de type entier ;
2. déclare les variables *a*, *b*, *c* et *d* de type réel ;
3. affecte la valeur 2 à *x* et 4 à *y* ;
4. calcule la somme, la différence, le produit, et le quotient de *x* par *y*, et affecte les résultats à *a*, *b*, *c* et *d* ;
5. affiche les valeurs de *a*, *b*, *c* et *d*.

Compilez et exécutez le programme.

Vous devez constater que le quotient de *x* par *y* (c'est-à-dire `x / y`) donne un résultat nul.

Modifiez ensuite votre programme en déclarant *x* et *y* avec le type *réel*.
Recompilez et exécutez.

Que constatez-vous ?

Même question avec *x* de type entier et *y* de type réel.
(Expliquez [à votre voisin] les différents résultats obtenus.)

Dernière mise à jour le 20 septembre 2017
Last modified: Wed Sep 20, 2017