# TP1: Utilisation du terminal et premiers programmes en C

Guillaume Rousseau
MP2I Lycée Pierre de Fermat
guillaume.rousseau@ens-lyon.fr

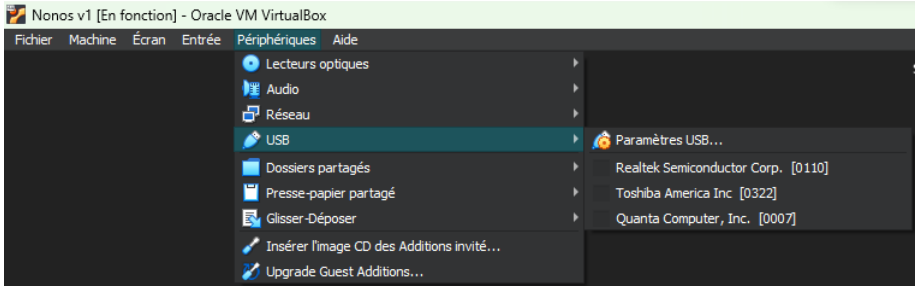
Bienvenue dans ce premier TP de C ! Avant de commencer le TP en lui même, quelques exercices pour apprendre à utiliser un ordinateur via le terminal, c'est à dire via des commandes textuelles, et en n'utilisant que très peu la souris. C'est une manière assez différente d'utiliser un ordinateur par rapport à ce dont vous pouvez avoir l'habitude, mais vous verrez qu'une fois qu'on a bien pris l'habitude c'est très rapide et très efficace. ***Lisez bien les parties suivantes*** car elles vous expliqueront entre autres comment rendre vos TP.

## Machine virtuelle

Les ordinateurs du lycée sont sous Windows, mais la programmation en C (et en Ocaml plus tard) se fait plus simplement sous Linux, et les épreuves de concours se déroulent également sous ce système d'exploitation. Le logiciel VirtualBox permet de simuler des machines sous n'importe quel système d'exploitation, ce que l'on appelle des **machines virtuelles**. Lorsque vous ouvrez VirtualBox, vous pouvez lancer la machine virtuelle "Nonos". Vous vous connectez alors à la session "eleve", avec le mot de passe "info".

Pour installer la machine virtuelle sur votre ordinateur personnel avec le même environnement que sur les ordinateurs du lycée, suivez les instructions dans le dossier que vous avez récupéré au premier cours.

**ATTENTION** Vous n'avez pas de session personnelle sur les machines virtuelles au lycée. Si vous y laissez un fichier, rien ne garantit que vous le retrouverez à la session suivante. Donc, à la fin de chaque TP, mettez votre travail sur une clé USB ou envoyez-le vous par mail. Lorsque vous branchez votre clé USB, elle se connecte par défaut à Windows et pas à la machine virtuelle. Il faut aller dans le menu "Périphériques" de VirtualBox et sélectionner votre clé dans le sous-menu "USB" :



## A Dossiers

Commençons par un peu de vocabulaire :

**Répertoires** Les ***répertoires***, ou ***dossiers*** sont des collections de fichiers. Les ordinateurs sont organisés en dossiers, avec une ***structure arborescente***, ce qui signifie que votre ordinateur contient des dossiers, qui eux même contiennent des dossiers, etc...

Il est possible de se déplacer dans l'ordinateur avec un explorateur de fichiers classique comme dans Windows ou MacOS. Dans "Applications", sélectionnez "Gestionnaire de fichiers" pour le lancer. Il y a également un raccourci sur la barre du bas.

# Exercice 1.

Lancez le gestionnaire de fichier. Vous vous trouvez par défaut dans /home/eleve. Vous devriez voir de nombreux fichiers et dossiers, dont certains ont un nom qui commencent par un point. Ces derniers sont des fichiers cachés dont vous n'avez pas besoin pour le moment, vous pouvez choisir de les voir ou pas en appuyant sur CTRL+H.

## B Navigation dans le terminal

Dans les TP, on travaillera principalement dans un terminal de l'ordinateur. Cela signifie que l'on interagit avec l'ordinateur en tapant des commandes et pas en cliquant à la souris dans une interface graphique.

# Exercice 2.

Ouvrez un terminal en cliquant sur l'icone  dans la barre des raccourcis en bas de votre écran. Vous pouvez aussi cliquer sur "Applications" en haut à gauche pour y trouver le terminal (ainsi que toutes les autres applications). Fermez la fenêtre du terminal, si un message d'erreur s'affiche, sélectionnez "ne plus afficher".

Dans un terminal, on navigue de dossier en dossier, et lorsque l'on ouvre un terminal, on se situe par défaut dans le répertoire ***/home/eleve*** : c'est la "page d'accueil" du système de fichier. Le répertoire où l'on se trouve lorsque l'on utilise un terminal s'appelle le **répertoire courant**, et il s'affiche sur chaque ligne à gauche de l'invite de commande :

Voyons quelques commandes utiles du terminal :

- Pour afficher le contenu actuel du dossier, on utilise la commande ***ls***.
- Pour se déplacer vers un autre dossier, on utilise la commande ***cd***. Par exemple, si l'on se trouve dans le dossier "parent", qui contient un sous dossier "parent/enfant", on peut accéder au sous-dossier en tapant :

  ```
  cd enfant
  ```

  Puis, pour revenir en arrière, on utilise la commande :

  ```
  cd ..
  ```

  En fait ".." signifie "Le répertoire qui me contient" dans le langage du terminal.
- Pour créer un dossier, on utilise la commande ***mkdir*** suivi du nom du dossier à créer.
- Pour créer un fichier, on utilise la commande ***touch*** suivi du nom du fichier à créer.

# Exercice 3.

Lancez un terminal puis utilisez des commandes pour répondre aux requêtes suivantes :

**Question 1.** Créez un répertoire appelé "TP1" puis accédez-y.

**Question 2.** Créez à l'intérieur de ce répertoire un sous-répertoire appelé "terminal", accédez y.

**Question 3.** Créez un fichier texte appelé ***blabla.txt***.

**Question 4.** Affichez le contenu du dossier avec ls et vérifiez que le fichier a bien été créé.

**Question 5.** Lancez la commande ***ls -a***. Que voyez-vous ? A votre avis, que signifie "-a" (indice : c'est l'initiale d'un mot anglais) ? Quels sont les deux dossiers qui s'affichent ici et qui ne s'affichaient pas avec ls ?

## C Création d'archive

Cette section va vous apprendre à créer une archive. Une archive permet de regrouper plusieurs fichiers et dossiers dans un seul gros fichier, afin de pouvoir facilement l'envoyer. Vous utiliserez cela pour rendre vos TP.

# Exercice 4.

1. Ouvrez un terminal, et placez vous là où vous avez créé le dossier ***TP1*** (si vous avez bien suivi les instructions, c'est dans "/home/eleve", c'est à dire la page d'accueil). Tapez :

   ```
   zip mon_archive.zip -r TP1
   ```

   Affichez le contenu du répertoire avec ls : que voyez vous ?
2. La commande permettant de décompresser une archive est ***unzip***. Créez un nouveau répertoire, appelé ***cible***, puis tapez dans le terminal :

   ```
   unzip mon__archive.zip -d cible
   ```

   Vérifiez que le répertoire cible contient maintenant le contenu de l'archive.

Alternativement, dans l'explorateur de fichiers, vous pouvez faire clic-droit puis "créer une archive" sur le dossier que vous voulez archiver.

## D Rendu de TP

Avant de continuer le TP, voici les instructions concernant le rendu :

— Chaque TP doit être rendu à la date indiquée avant 22h00, en déposant votre rendu sur le cahier de prépa de la classe, dans la section ***Informatique***, sur la page ***Transfert de documents***.

— Votre rendu doit être composé d'une archive (c'est à dire un .zip) contenant votre code et un compte rendu écrit avec les réponses des questions du TP.

— Plus précisément, vous devrez donc créer une archive à partir du répertoire ***TP1***, que vous appellerez ***TP1_nom_de_famille.zip***.

— Chaque TP a plusieurs exercices, chaque exercice ayant plusieurs questions. Vous créerez un sous-dossier par exercice, et vous y mettrez le code que vous aurez produit pour cet exercice.

— L'archive devra également contenir un fichier "reponses.txt" contenant les éventuelles réponses écrites, ainsi que des remarques (une partie du TP que vous n'avez pas réussi, ou bien un détail sur lequel vous avez une question, etc...).

— Merci de bien respecter les consignes, car avoir tous vos rendus sous la même forme m'aide à gagner beaucoup de temps et à automatiser ce qui peut l'être : vous êtes plus de 40 dans la classe, ça représente beaucoup de travail de vous corriger !

Travaillez les TP régulièrement afin de ne pas devoir travailler en panique le soir de la date de rendu et de pouvoir rendre le TP en avance. En effet :

**Théorème 1.** Le plus tôt vous rendez un TP/DM, le plus tôt je peux le corriger et vous aider à progresser.

Vous êtes évidemment encouragé·e·s à m'envoyer un mail ou à venir me voir avant/après les cours si vous avez une question ou que vous bloquez sur un TP.

## E Édition de texte

Pour écrire du code, on utilise un éditeur de texte. Sur la virtuelle, il y a plusieurs choix d'éditeurs de texte. Les trois suivants sont installés sur les machines virtuelles, et sont accessible dans la barre des raccourcis en haut de l'écran :

— ***VSCodium***, dans ***Applications - Développement - VSCodium***, est un editeur de code assez puissant, avec un système d'autocomplétion et d'autres fonctionnalités adaptées au développement.

— ***gedit***, dans ***Applications - Accessoires - Text Editor***, est un éditeur de texte basique.

Pour ouvrir un fichier dans VSCodium directement depuis le terminal, il vous suffit de taper `codium nom_du_fichier`. Pour gedit, tapez `gedit nom_du_fichier`. Si le fichier a une extension comme .c (fichiers C), .py (fichiers python), .ml (fichiers OCaml), etc... alors l'éditeur active automatiquement la coloration syntaxique du langage concerné.

**Exercice 5.** Dans un terminal, créez un fichier "prog.py", puis lancez VSCodium et tapez deux lignes de python dans ce fichier.

## F Premier programme C

Dans un programme C, il y a plusieurs choses à faire avant même de commencer à écrire le code à proprement parler :

**Librairies** Par défaut, un programme C ne peut pas faire grand chose. Presque toutes les fonctionnalités se situent dans des librairies, c'est à dire dans des bouts de code C spécialisés que l'on peut inclure dans son propre code. Par exemple, la fonction la plus simple pour afficher quelque chose dans le terminal est ***printf***. Pour pouvoir l'utiliser, on a besoin de la librairie ***stdio.h***, signifiant "**st**andar**d** **i**n/**o**ut" (soit entrée/sortie standard). Pour déclarer que l'on utilise cette librairie, on commence notre programme par :

```
1 #include <stdio.h>
```

**La fonction main** Une fonction est un bloc de code ayant une utilité précise auquel on a donné un nom, souvent dans le but de le réutiliser. Tout comme un algorithme, une fonction a des entrées et une sortie (que l'on appelle parfois valeur de retour). Un programme C peut comporter des dizaines, des centaines, des milliers de fonctions. Pour l'instant, on n'en utilise qu'une seule : la fonction ***main***. La fonction ***main*** ("principale" en anglais) est celle qui va être lancée lorsque l'on exécute le programme compilé. Si aucune fonction ne s'appelle main, le compilateur va renvoyer une erreur, car il ne saura pas où faire commencer le programme. **Un programme C DOIT avoir une fonction main !**

La fonction main ***retourne*** un entier, qui est un code indiquant si le programme s'est déroulé normalement. Par convention, la valeur indiquant que tout va bien est 0. Ainsi, pour l'instant, nos programmes ressembleront à :

```c
#include <stdio.h>

int main(){
  /*
  Votre code ici
  */
  return 0;
}
```

Lorsque vous créez un nouveau fichier C, vous pouvez donc toujours commencer par taper ces lignes, et remplir les trous après.

**Remarque.** Si une ligne commence par //, son contenu n'est pas pris en compte lors de la compilation. De même, tout ce qui est écrit entre /* et */ n'est pas pris en compte, y compris sur plusieurs lignes. On appelle ces lignes des ***commentaire***. Les commentaires permettent de donner des informations aux personnes qui lisent votre code, de vous organiser lorsque vous écrivez un programme, de cacher un bout de code pour tester des alternatives, etc... Bien commenter son code est ***NECESSAIRE*** pour être un·e bon·ne informaticien·ne ! On verra au fil des TPs comment utiliser les commentaires à bon escient.

**C'est parti** La tradition veut que lorsque l'on apprend un nouveau langage, le premier programme que l'on écrit affiche "Hello world".

## Exercice 6.

**Question 1.** Créez un fichier **hello_world.c**, et à l'aide d'un éditeur de texte, recopiez-y le code C suivant :

```c
#include<stdio.h>

int main(){
  // Afficher une salutation
  printf("Hello world");
  return 0;
}
```

Il ne reste plus qu'à apprendre comment **compiler** ce programme, c'est à dire le traduire en langage machine.

## G Compilation de code C

Le C est un langage compilé : on a besoin d'un programme pour traduire le code C en code machine, c'est à dire d'un compilateur. Le compilateur le plus connu pour le C s'appelle ***GCC***, ce qui signifie ***GNU C Compiler*** (GNU étant un des premiers système d'exploitation).

Pour compiler un fichier, il suffit de lancer le compilateur sur ce programme. Par exemple, pour compiler le fichier ***mon_programme.c***, on tape dans le terminal :

**gcc mon_programme.c**

Attention, il faut que vous soyez dans le répertoire où se trouve le fichier à compiler pour que cette commande marche !

Cette commande va créer un nouveau fichier, le résultat de la compilation, qui n'est plus lisible facilement pour les humains, mais que la machine peut exécuter. On appelle un tel fichier un ***exécutable*** ou un ***binaire***. Tous les programmes que l'on utilise dans la vraie vie (Chrome, PowerPoint, Fortnite, Spotify, etc...) sont des exécutables !

```c
#include <stdio.h>

int main()
{
  printf("Bonjour\n");
  return 0;
}
```

FIGURE 1 – mon_programme.c

```
guillaume@MSI:~/demo_tp_c/compil$ ls
mon_programme.c
guillaume@MSI:~/demo_tp_c/compil$ gcc mon_programme.c
guillaume@MSI:~/demo_tp_c/compil$ ls
a.out  mon_programme.c
guillaume@MSI:~/demo_tp_c/compil$ ./a.out
Bonjour
guillaume@MSI:~/demo_tp_c/compil$
```

FIGURE 2 – Compilation et exécution de mon_programme.c

**Remarque.** Sans options supplémentaires, le programme exécutable compilé s'appelle ***a.out***. On peut spécifier un nom on utilisant l'option ***-o*** (comme ***o***utput). On tape donc :

**gcc fiercher_source.c -o nom_executable**

```
guillaume@MSI:~/demo_tp_c/compil_nom$ ls
mon_programme.c
guillaume@MSI:~/demo_tp_c/compil_nom$ gcc mon_programme.c -o affiche_bonjour
guillaume@MSI:~/demo_tp_c/compil_nom$ ls
affiche_bonjour  mon_programme.c
guillaume@MSI:~/demo_tp_c/compil_nom$ ./affiche_bonjour
Bonjour
guillaume@MSI:~/demo_tp_c/compil_nom$
```

FIGURE 3 – Compilation et exécution de mon_programme.c en affiche_bonjour

Remarquez qu'une fois que le programme affiche_bonjour a été créé, pour l'exécuter il faut taper **./affiche_bonjour** et pas seulement **affiche_bonjour** !

# Exercice 7.

**Question 1.** Compilez le fichier "hello_world.c", en nommant l'exécutable ***hello_world***.

**Question 2.** Lancez l'exécutable en tapant ***./hello_world***. Que remarquez-vous au niveau de l'affichage ?

Pour corriger le problème d'affichage, nous allons rajouter un retour à la ligne. En C, et dans la plupart des langages, le retour à la ligne s'écrit ***"\n"*** (comme ***n***ewline), et s'insère directement dans le texte à afficher.

**Question 3.** Modifiez votre code pour rajouter un retour à la ligne au texte affiché et relancez votre programme. N'oubliez pas de recompiler avant !

**Question 4.** Rajoutez un commentaire multiligne au dessus de la fonction main avec votre prénom, nom, et la date d'aujourd'hui.

**Remarque 1.** Un programme C qui n'est pas compilé ne sert pas à grand chose. Compiler et exécuter est le seul moyen de pouvoir tester votre programme. Pensez donc à compiler et tester le code que vous écrivez régulièrement, et ne rendez jamais de code que vous n'avez pas essayé de compiler !

## H Un langage impératif

Le C est un langage impératif, ce qui signifie qu'un programme C est une suite d'instructions que l'on donne à la machine. Par exemple, dans le programme que vous venez d'écrire, `printf("Hello world !\n");` est une instruction.

Un langage impératif utilise également des ***variables***, c'est à dire des boîtes dans lesquelles on peut stocker des valeurs afin de les réutiliser. Par exemple, en C, si l'on écrit :

```
1 x = 5;
```

on stocke le nombre 5 dans la variable appelée $x$. Ainsi, si plus tard dans le code on écrit $2 * x$, on trouvera bien 10. On peut évidemment changer la valeur stockée dans une variable plusieurs fois :

```
1 x = 5;
2 x = 2*x;
3 x = 2*x*x - 5*x + 1;
```

## I Un langage typé

Le C est un langage typé, ce qui signifie que toutes les variables et les valeurs que l'on y manipule ont un type. Ce type doit être nécessairement indiqué lorsque l'on crée une variable : il indique au compilateur la taille nécessaire au stockage des données et lui permet aussi de détecter certaines erreurs. Voici deux types de base du C :

- ***int*** : c'est le type des entiers. Il permet de représenter des nombres entre -2 147 483 648 et 2 147 483 647, c'est à dire entre $-2^{31}$ et $2^{31} - 1$, et tient donc sur 32 bits.
- ***float*** : c'est le type qui se rapproche le plus des nombres réels. Il signifie "nombre à virgule flottante", ou ***nombre flottant*** par abus de langage. Il tient également sur 32 bits.

Pour créer une variable entière s'appelant *coco* par exemple, on écrit :

```
1 int coco;
```

On appelle cela une ***déclaration de variable***. On dit que l'on a déclaré *coco*. Lors d'une déclaration, on peut directement assigner une valeur initiale à la variable. Ainsi, les deux codes suivants se comportent pareil :

```
1 int x = 5;
```

```
1 int x;
2 x = 5;
```

## Exercice 8.

Sur Cahier de Prépa, dans les documents à télécharger, vous trouverez une archive pour le TP1. Cette archive contient un fichier "a_completer.c" contenant le code suivant :

```c
#include <stdio.h>

int main(){
  // créer une variable x, initialement nulle
  int x =

  // créer une variable y, valant 5 initialement

  // modifier x en lui assignant x+y

  // modifier y en lui assignant y+x

  // répéter les deux dernières étapes une fois de plus

  // afficher x et y
  printf("x vaut %d, y vaut %d\n", x, y);
  return 0;
}
```

**Question 1.** Décompressez l'archive et copiez le fichier dans votre répertoire de TP (dans le sous-dossier correspondant à cet exercice).

**Question 2.** Complétez ce programme en suivant les requêtes des commentaires.

**Question 3.** Compilez et exécutez votre programme, et vérifiez qu'il affiche "x vaut 15, y vaut 25".

Vous remarquerez que chaque ligne se termine par un point-virgule. En C, les retours à la ligne ne comptent pas, et la différence entre les différentes parties du programme est faîte grâce aux accolades { et }, et aux points virgules. Même si l'on peut techniquement écrire tout programme C sur une seule ligne, on s'assure en pratique de faire un retour à la ligne entre chaque instruction, et de sauter des lignes entre les différents blocs de code.

**Remarque.** Il existe des compétitions sur internet pour écrire des programmes en utilisant le moins de caractères possible. Le cours d'informatique n'en est **pas** une.

## J Manipulation des nombres

Commençons à écrire quelques programmes qui utilisent les types ***int*** et ***float***. Nous avons vu qu'en C, les variables doivent être ***déclarées*** ainsi que leur type. Par exemple, pour créer une variable $x$ entière valant 5, on écrit :

```c
int x = 5;
```

De même, pour créer une variable *zouzou* flottante qui vaut 6.549, on écrit :

```c
float zouzou = 6.549;
```

Pour afficher les entiers et les flottants, on utilise à nouveau la fonction ***printf***. La syntaxe est un peu particulière : on utilise "%d" ou "%f" pour indiquer dans le texte que l'on souhaite insérer un entier ou un flottant, respectivement, puis on spécifie les nombres à afficher, dans le même ordre.

# Exercice 9.

Lisez le code suivant :

```c
#include <stdio.h>

int main(){
  int n = 15 ;
  float prix = 6.99 ;

  printf("J'ai acheté %d t-shirts à %f euros\n", n, prix);

  return 0;
}
```

1. Qu'afficherait le programme une fois compilé et exécuté ?
2. Créez un nouveau fichier C, recopiez le programme ci-dessus, compilez et exécutez. Le résultat est-il bien ce que vous aviez prévu ?

Le type d'une variable permet d'indiquer son comportement lié à certaines opérations. Par exemple, la division n'aura pas le même effet sur les ***int*** et les ***float***.

# Exercice 10.

**Question 1.** Créez un nouveau fichier C.

**Question 2.** Créez deux variables flottantes x et y valant respectivement 5 et 2.

**Question 3.** Créez une nouvelle variable flottante z contenant $\frac{x}{y}$ (en C, la division se fait grâce au symbole /). Affichez la valeur de cette variable.

**Question 4.** Compilez et exécutez le programme, et vérifiez qu'il affiche bien 2.5.

**Question 5.** Recommencez en utilisant des variables entières à la place.

**Question 6.** Compilez et exécutez le programme : que remarquez vous ? Pourquoi ?

Sur les entiers et les flottants, on peut donc utiliser les opérations mathématiques classiques ($+$, $-$, $\times$, $/$) comme sur une calculatrice. Le symbole pour la multiplication est l'astérisque $*$.

# Exercice 11.

Essayons de mélanger les types. Créez un nouveau fichier C.

**Question 1.** Déclarez une variable x entière, valant 3

**Question 2.** Déclarez une variable y flottante, valant 0.25

**Question 3.** Déclarez une variable somme_f flottante valant $x + y$

**Question 4.** Déclarez une variable somme_i entière valant $x + y$

**Question 5.** Affichez le contenu des 4 variables. Que remarquez vous ?

## K Interaction avec le terminal

La fonction printf permet aux programmes d'afficher des informations formatées, d'où son nom : "print" signifie "afficher", et le f signifie "formaté".

La fonction ***scanf*** permet de lire dans le terminal. Elle s'utilise de manière similaire à printf en précisant avec "%d" et "%f" ce qu'il faut scanner.

# Exercice 12.

**Question 1.** Lisez le code suivant :

```
1  #include <stdio.h>
2
3  int main(){
4    int x, y; // permet de déclarer deux variables en mÃªme temps
5
6    printf("Entrez x: ");
7    scanf("%d", &x);
8
9    printf("Entrez y: ");
10   scanf("%d", &y);
11
12   printf("x vaut %d, y vaut %d, la somme vaut %d\n", x, y, x+y);
13   return 0;
14 }
```

Que fait-il ?

**Question 2.** Recopiez ce code dans un nouveau fichier C et exécutez le. Testez avec plusieurs valeurs de x et y.

**Question 3.** Essayez de rentrer autre chose que des nombres pour faire bugger le programme. Que se passe-t'il ?

**Remarque.** Un programme qui bug peut avoir un comportement totalement inattendu, et peut avoir des comportements différents d'une exécution à l'autre !

# Exercice 13.

**Question 1.** Recopiez et exécutez le programme suivant :

```
1  #include <stdio.h>
2
3  int main(){
4    int x, y, z;
5    printf("Entrez des valeurs pour x, y et z:\n");
6    scanf("%d %d %d", &x, &y, &z);
7    printf("La somme des trois nombres est %d\n", x+y+z);
8    return 0;
9  }
```

**Question 2.** Modifier le programme précédent pour qu'il fasse la multiplication de deux flottants plutôt que la somme de trois réels.

On remarque que dans les arguments de scanf, lorsque l'on donne les variables où stocker les nombres lus, on les précède par le signe &. Ce symbole, que nous étudierons plus en détail au chapitre 3, permet d'obtenir l'adresse d'une variable, c'est à dire son emplacement dans la mémoire de l'ordinateur.

## L Conditions

On rajoute de nouvelles instructions qui permettent de complexifier nos programmes. Ces instructions vont permettre de tester des conditions, et d'exécuter différentes parties du code selon si une condition est vérifiée ou pas.

# Exercice 14.

Lisez le code suivant :

```c
#include <stdio.h>

int main()
{
  int x;
  int y;

  printf("Entrez x: ");
  scanf("%d", &x);
  printf("Entrez y: ");
  scanf("%d", &y);

  if (x < y){
    printf("oui\n");
  } else {
    printf("non\n");
  }

  return 0;
}
```

Que veut dire "if" en anglais ? Et "else" ? A votre avis, que fait ce programme ?

Ce nouvel élément du langage s'appelle ***instruction conditionnelle***, ou ***if-else***, ou encore ***if-then-else***. La syntaxe est simple :

```c
if (CONDITION){
  /*
  CODE SI VRAI
  */
} else {
  /*
  CODE SI FAUX
  */
}
```

On appelle les morceaux de code à exécuter dans un cas ou dans l'autre les ***branches***.

Il est même possible de ne pas spécifier de code à exécuter lorsque la condition est fausse, dans ce cas la syntaxe est :

```c
if (CONDITION){
  /*
  CODE
  */
}
```

Il existe plusieurs types de conditions :

- Le test d'égalité s'écrit `x == y`
- Le test d'inégalité s'écrit `x != y`
- Le test d'infériorité stricte ("$<$") s'écrit `x < y`
- Le test d'infériorité large ("$\leq$") s'écrit `x <= y`
- Le test de supériorité stricte ("$>$") s'écrit `x > y`
- Le test de supériorité large ("$\geq$") s'écrit `x >= y`

# Exercice 15.

Écrivez les programmes suivants (il faut donc un fichier C par question) et testez les sur plusieurs entrées :

1. Un programme demandant de rentrer un entier, et qui dit s'il vaut 0 ou pas.
2. Un programme demandant de rentrer un nombre, et qui dit "cas A", "cas B" ou "cas C" selon si le nombre est :
   - A Strictement plus petit que -0.33
   - B Entre -0.33 et 7.89 au sens large
   - C Strictement plus grand que 7.89

Afin d'éviter de faire des if imbriqués, on peut combiner les conditions. Il existe 3 opérateurs permettant de faire ces combinaisons :

— Le "ou", noté ||. Par exemple : `x = 0 || x = 3` sera vérifiée si, et seulement si, l'une des deux conditions au moins est vérifiée, donc si x vaut 0 ou 3.

— Le "et", noté &&. Par exemple : `0 < x && x < 3` sera vérifiée si, et seulement si, les deux conditions sont vérifiées, donc si x est strictement compris entre 0 et 3.

— Le "non", noté !. Par exemple : `!(0 < x && x < 3)` sera vérifiée si, et seulement si, la condition intérieure ne l'est pas, donc si et seulement si x n'est pas strictement contenu entre 0 et 3.

# Exercice 16.

Utilisez ces opérateurs pour écrire les programmes suivants :

1. Un programme demandant de rentrer trois nombres et qui affiche celui qui est entre les deux autres (i.e. la médiane).
2. Un programme demandant de rentrer un entier, puis affiche "gou" si c'est un multiple de 3, puis affiche "ba" si c'est un multiple de 5, et affiche le nombre seulement s'il n'est multiple ni de 3 ni de 5 (donc si vous rentrez 15 le programme affiche "gouba", si vous rentrez 14 il affiche 14). ***Aide*** : `x % 3` permet d'obtenir le reste modulo 3 de $x$.

Testez bien vos programmes sur des valeurs variées. En particulier pour le deuxième, vérifiez que dans tous les cas le retour à la ligne s'effectue comme il faut.

## M Gestion du temps et de l'aléatoire

# Exercice 17.

La librairie **time.h** fournit des fonctions en rapport avec le temps. En particulier, la fonction ***time*** donne la date actuelle. Cependant elle la donne sous un format particulier : elle donne le nombre de secondes écoulées depuis le ***1er janvier 1970 à 00h00m00s***. Pour l'utiliser, on écrit :

```
1 t = time(NULL);
```

ne vous préoccupez pas de ce que veut dire NULL pour le moment, on verra ça plus tard.

**Question 1.** Ce nombre peut-il tenir dans une variable de type int ? En quelle année environ dépasserait-t-il la limite sur un int (voir partie I) ?

Sur la plupart des machines modernes, la fonction time ne renvoie en fait pas un int, mais un ***long int*** de 64 bits, qui peut représenter les nombres de l'intervalle $[\![-2^{63}, 2^{63}-1]\!]$. C'est le cas des machines du lycée. Il faut donc stocker le résultat dans une variable du même type. De plus, le format pour afficher ce type est %ld au lieu de %d :

```
1 long int t;
2 t = time(NULL);
3 printf("Le nombre de secondes depuis le 01/01/1970 est %ld\n", t);
```

**Question 2.** Jusqu'à quelle date environ peut-on utiliser ce format ?

**Question 3.** En utilisant la fonction time, écrivez un programme qui donne l'année et le mois actuel. ***Aide*** *: une année fait environ 365.24 jours en moyenne, un mois fait environ 30.44 jours en moyenne.*

# Exercice 18.

La librairie **stdlib.h** contient de nombreux outils, dont des fonctions permettant de générer des nombres aléatoires. Dans un programme utilisant de l'aléatoire, il faut initialiser le générateur de nombre en lui fournissant une ***seed*** (ça parlera aux joueur·euse·s de Minecraft). Une bonne solution est de choisir comme seed la valeur renvoyée par la fonction time car elle change souvent (une fois par seconde).

**Question 1.** Recopiez, et compilez le code suivant :

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <time.h>
4
5  int main()
6  {
7    // choix de la seed pour l'aléatoire
8    srand(time(NULL));
9
10   int x = rand();
11   int y = rand();
12
13   printf("x: %d, y: %d\n", x, y);
14
15   return 0;
16 }
```

Exécutez maintenant plusieurs fois le programme compilé, à quelques secondes d'intervalle. Qu'observez-vous ?

**Question 2.** Exécutez le programme compilé plusieurs fois en une seconde. Que constatez-vous ? (***Aide** : la flèche du haut permet de remonter l'historique du terminal, vous pouvez donc répéter rapidement une commande en faisant HAUT puis ENTRÉE.*)

Pour générer un nombre aléatoire entre 0 et $k-1$ avec $k \in \mathbb{N}^*$, on prend le résultat de rand() modulo $k$. On considère que le nombre obtenu est choisi de manière uniforme (mais en réalité ce n'est pas exactement le cas).

**Question 3.** Écrivez un programme qui tire et affiche un nombre aléatoire entre 0 et 99 ;

**Question 4.** Écrivez un programme qui tire et affiche trois nombres aléatoires entre 10 et 20 ;

## N Exercice libre

# Exercice 19.

Imaginez et implémentez un programme qui utilise les notions suivants :

— Les types int/float

— Des opérations arithmétiques

— Un ou plusieurs if-else

— De la saisie et de l'affichage

Commentez ce programme pour expliquer succinctement ce qu'il fait, en segmentant les étapes principales, en décrivant les variables cruciales, etc... Si vous n'avez pas d'idées, vous pouvez piocher dans la liste suivante :

— Un comparateur de prix : on saisit le poids en kilogrammes et le prix en euros de deux articles, et le programme dit lequel coûte le moins cher au kilo.

— Un programme demandant de rentrer un chiffre entre 1 et 9, et qui affiche ce chiffre en toutes lettres.

— Un programme calculant les solutions d'une équation du second degré

— Un jeu demandant de résoudre une grosse addition, et qui affiche le temps mis pour la résoudre.

— Un quiz chronométré sur les trois œuvres du programme de français.

Si vous êtes déjà à l'aise en C, vous pouvez évidemment utiliser des notions qu'on n'a pas encore vu : fonctions, boucles, pointeurs, structures, fichiers...