

TP1: Utilisation du terminal et premiers programmes en C

MP2I Lycée Pierre de Fermat
guillaume.rousseau@ac-toulouse.fr

Bienvenue dans ce premier TP de C ! Avant de commencer à coder en C, quelques exercices pour apprendre à utiliser un ordinateur via le terminal, c'est à dire via des commandes tapées au clavier, en n'utilisant presque pas 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'avec un peu d'entraînement c'est bien pratique.

Débian

Les ordinateurs du lycée sont sous Windows, mais la programmation en C (et en Ocaml plus tard) se fait plus simplement sous la famille de systèmes d'exploitation appelée **Unix**, qui est aussi celle utilisée lors des concours. Lorsque vous allumez les ordinateurs des salles, vous tombez sur un menu de démarrage appelé GRUB qui permet de choisir entre Windows et Debian (un système de la famille Unix). Attention, sans action de votre part, **GRUB lance Windows automatiquement** au bout de quelques secondes.

Exercice 1

Allumez votre ordinateur, et dans GRUB, sélectionner Debian. Lorsque vous atteignez la page d'accueil, connectez vous avec le nom d'utilisateur "eleve", et le mot de passe "info".

Travailler à la maison

Pour travailler chez vous, si vous avez un ordinateur sous Linux ou Mac, vous n'avez rien à faire, vous disposez déjà d'un environnement similaire à celui du lycée.

Si vous êtes sur Windows, vous devez **installer un dual boot** sur votre ordinateur.

Sauvegarder son travail

ATTENTION, vous n'avez pas de session personnelle sur les Debian du lycée. Si vous y laissez un fichier, aucune garantie 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.

A Dossiers

Les **dossiers**, ou **répertoires** 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 sur Windows ou Mac, avec la souris. Pour cela, cliquez sur "Applications", en haut à gauche, puis sélectionnez "Gestionnaire de fichiers" pour le lancer. Il y a également un raccourci sur la barre du bas, en forme de meuble à tiroirs.

Exercice 2

Lancez le gestionnaire de fichier. Vous vous trouvez par défaut dans le dossier `/home/eleve`. Vous devriez voir quelques fichiers ou dossiers. Dans Debian, les fichiers dont le nom commence par un point sont considérés comme des fichiers cachés. Vous pouvez choisir de les voir ou pas en appuyant sur CTRL+H.

Exercice 3

Créer un fichier sur le bureau en faisant clic-droit. Brancher une clé USB, l'ouvrir en utilisant l'explorateur de fichier, et transférer le fichier créé sur votre clé.

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 4

Ouvrez un terminal en cliquant sur l'icône ■ 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 le terminal.

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 compte "eleve" dans le système de fichier. Le dossier où l'on se trouve lorsque l'on utilise un terminal s'appelle le **répertoire courant** (répertoire et dossier veulent dire la même chose), et il s'affiche sur chaque ligne à gauche de l'invite de commande.

Voyons quelques commandes utiles du terminal:

- **ls** affiche le contenu du répertoire courant.
- **cd** suivi d'un nom de dossier permet de se déplacer vers ce dossier. C'est comme double-cliquer sur un dossier dans l'explorateur pour rentrer à l'intérieur. Taper **cd ..** permet de revenir en arrière (les deux points font partie de la commande).
- **mkdir** suivi d'un nom de dossier permet de créer ce dossier.
- **touch** suivi d'un nom de fichier permet de créer ce fichier. Attention, il faut toujours spécifier l'extension du fichier que l'on crée: **.txt**, **.c**, **.py**, etc...
- **thunar** permet d'ouvrir l'explorateur de fichiers. Par défaut, il s'ouvre au niveau du répertoire personnel de l'utilisateur. On peut spécifier **thunar .** (le point fait partie de la commande) pour ouvrir l'explorateur au niveau du répertoire courant.

Dans le langage du terminal, **.** signifie "le répertoire courant", et **..** signifie "Le répertoire qui contient le répertoire courant".

Exercice 5

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

- Q1.** Créez un dossier appelé **TP1**, contenant un sous-dossier appelé **terminal**, et placez-vous dans ce dernier.
- Q2.** Créez un fichier texte appelé **blabla.txt**, et affichez le contenu du dossier pour vérifier que le fichier a bien été créé.
- Q3.** Créez un fichier texte appelé **.secret.txt**, et relancez **ls**: que constate t-on ?
- Q4.** Lancez la commande **ls -a**. A votre avis, que signifie **"-a"** ?
- Q5.** Tapez **ls --help** pour afficher le mode d'emploi de la commande **ls**, et cherchez ce que fait précisément l'option **-a**.
- Q6.** Ouvrez un explorateur de fichier à l'endroit où vous vous trouvez.

C Création d'archive

Une archive permet de regrouper plusieurs fichiers et dossiers dans un seul gros fichier, afin de pouvoir facilement l'envoyer. Vous rendrez systématiquement vos TP sous ce format.

Exercice 6

- Q1.** Ouvrez un terminal, et placez vous là où vous avez créé le dossier **TP1** (c'est à dire à l'extérieur du dossier **TP1**). Vérifiez avec **ls** que **TP1** apparaît bien.
- Q2.** Tapez **zip mon_archive.zip -r TP1** puis affichez le contenu du répertoire courant avec **ls**. Le nouveau fichier apparu est une **archive** du dossier.
- Q3.** Créez un nouveau répertoire, appelé **cible**, puis tapez dans le terminal la commande **unzip mon_archive.zip -d cible**, et vérifiez que le répertoire **cible** contient maintenant le contenu de l'archive.

A retenir La commande `zip` permet d'archiver un dossier, et la commande `unzip` permet de décompresser une archive pour la retransformer en un dossier. 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 ce que vous devez rendre à l'issue de chaque TP:

- A la date indiquée sur le sujet ou donnée à l'oral, avant 21h00, vous devez déposer votre rendu sur le site cahier de prépa, dans la section **Informatique**, sur la page **Transfert de documents**. Tous les exercices sont à faire, mais seuls certains seront à rendre.
- Votre rendu doit être composé d'une archive (c'est à dire un .zip) contenant votre code et les réponses aux éventuelles questions écrites du TP.
- Lorsqu'un TP a plusieurs exercices, vous créerez **un sous-dossier par exercice**, dans lequel vous mettrez tout le code correspondant.
- L'archive devra également contenir un unique fichier "reponses.txt" **sans accent** (n'en parlez pas à votre prof de Lettres) contenant les réponses écrites, ainsi que d'éventuelles remarques que vous pouvez avoir (une partie du TP que vous n'avez pas réussi, 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, vous corriger représente beaucoup de travail !

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 et rectifier les erreurs.

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 code

Le logiciel VSCodium est un éditeur de code assez pratique, qui dispose de coloration syntaxique, d'autocomplétion, et de tout un tas d'autres fonctionnalités adaptées à l'écriture de programmes. Pour ouvrir un fichier dans VSCodium directement depuis le terminal, il vous suffit de taper la commande `codium nom_du_fichier`. Si le fichier a une extension comme .c (fichiers C), .py (fichiers python), .ml (fichiers OCaml), ou autre, l'éditeur active automatiquement la coloration syntaxique du langage concerné.

Exercice 7

Dans un terminal, créez un fichier `prog.py`, puis ouvrez le avec VSCodium et tapez deux lignes de python. Vérifiez que les mots-clés comme `def` sont en couleur.

F Premier programme C

Un programme C a une structure un peu plus complexe qu'un programme Python. En 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. Les différentes fonctionnalités sont stockées dans des fichiers appelés des librairies (très similaires aux modules Python tels que numpy, que vous avez sûrement déjà utilisé). Par exemple, la fonction la plus simple pour afficher du texte dans le terminal, `printf`, est stockée dans la librairie ***stdio.h***, signifiant "standard in/out" (entrée/sortie standard). Si l'on veut utiliser `printf`, il faut déclarer au début du programme que l'on utilise la librairie adaptée, avec la ligne:

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

La fonction main Une fonction est un bloc de code auquel on a donné un nom et qui, comme un algorithme, peut avoir des entrées et une sortie. Bien qu'un programme C peut comporter de nombreuses fonctions, on n'en utilise qu'une seule pour le moment: la fonction ***main*** ("principale" en anglais). Elle est lancée au moment où l'on exécute le programme, c'est son point de départ. Un programme C **doit** avoir une fonction main, sans quoi il ne peut pas se lancer.

Ainsi, pour l'instant, nos programmes ressembleront à:

```
1 #include <stdio.h>
2
3 int main(){
4     /*
5     Votre code ici
6     */
7     return 0;
8 }
```

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

La dernière ligne est la valeur de sortie de la fonction, aussi appelée sa valeur de retour. Pour la fonction main, c'est un entier indiquant comment s'est déroulé le programme. Par convention, la valeur indiquant que tout va bien est 0.

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

Exercice 8

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

```
1 #include<stdio.h>
2
3 int main(){
4     // Afficher une salutation
5     printf("Hello world");
6     return 0;
7 }
```

En C, tout ce qui est écrit entre `/*` et `*/` n'est pas pris en compte, y compris sur plusieurs lignes. Idem pour les lignes commençant par `//`. On appelle ces lignes des **commentaires**, elles permettent de donner toutes sortes d'informations aux personnes qui lisent votre code, y compris à vous-même. On verra au fil des TP comment faire de bons commentaires.

G Compilation de code C

Le programme que l'on a écrit n'est pas utilisable en tant que tel, car votre ordinateur n'est pas capable d'exécuter directement du code C. Un programme C doit d'abord être traduit en langage machine, par un logiciel appelé **compilateur**. Le compilateur le plus connu pour le C s'appelle **GCC**, ce qui signifie **GNU C Compiler**¹.

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 ² :

```
gcc mon_programme.c
```

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 **binnaire**. Tous les programmes que l'on utilise quotidiennement (Firefox, PowerPoint, Fortnite, Spotify, etc...) sont des exécutables !

```
1 #include <stdio.h>
2
3 int main(){
4     printf("Bonjour\n");
5     return 0;
6 }
```

```
eleve@ordi:~/TP1$ ls
mon_programme.c
eleve@ordi:~/TP1$ gcc mon_programme.c
eleve@ordi:~/TP1$ ls
a.out mon_programme.c
eleve@ordi:~/TP1$ ./a.out
Bonjour
```

Figure 1: `mon_programme.c`

Figure 2: Compilation et exécution

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

```
gcc fichier_source.c -o nom_executable
```

Une fois que le programme a été compilé, pour l'exécuter il faut taper `./nom_executable` et pas juste `nom_executable` !

Exercice 9

Q1. Compilez le fichier `hello_world.c`, en nommant l'exécutable `hello`, puis lancez cet exécutable.

On remarque qu'il manque un retour à la ligne. En C, le retour à la ligne s'écrit `"\n"` (comme **newline**), et s'insère directement dans le texte à afficher (voir Fig. 1).

Q2. Modifiez votre code pour rajouter un retour à la ligne au texte affiché et relancez votre programme. N'oubliez pas de recompiler !

¹GNU est un système d'exploitation libre de droits reprenant les concepts du système UNIX qui le précède. Je vous laisse chercher la signification de l'acronyme GNU.

²Attention, il faut que vous soyez dans le répertoire où se trouve le fichier à compiler !

Remarque 1. Un programme C qui n'est pas compilé ne sert à rien. Compiler et exécuter est le seul moyen de pouvoir tester votre programme, de voir et corriger les bugs, et mieux comprendre la programmation C. Pensez donc à compiler et tester le code que vous écrivez régulièrement, et **ne rendez jamais de code que vous n'avez pas compilé !**

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 modifier la valeur stockée dans une variable:

```
1 x = 5;
2 x = 2*x; // x vaut 10
3 x = 2*x*x - 5*x + 1; // x vaut 151
```

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 faite 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 **code-golf**). Le cours d'informatique n'en est **pas** une.

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 $2^{31} - 1$. On verra pourquoi en cours, et on verra que cela en fait un type 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 flottante s'appelant *coco* par exemple, on écrit:

```
1 float 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 bouts de code suivants se comportent pareil:

```
1 float x = 5.25;
```

```
1 float x;
2 x = 5.25;
```

Exercice 10

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:

```
1  #include <stdio.h>
2
3  int main(){
4      // créer une variable entière x, initialement nulle
5      int x =
6
7      // créer une variable entière y, valant 5 initialement
8
9      // modifier x en lui assignant x+y
10
11     // modifier y en lui assignant y+x
12
13     // répéter les deux dernières étapes une fois de plus
14
15     // afficher x et y
16     printf("x vaut %d, y vaut %d\n", x, y);
17     return 0;
18 }
```

- Q1.** Décompressez l'archive et copiez le fichier dans votre répertoire de TP (dans le sous-dossier correspondant à cet exercice).
- Q2.** Complétez ce programme en suivant les requêtes des commentaires.
- Q3.** Compilez et exécutez votre programme, et vérifiez qu'il affiche "x vaut 15, y vaut 25".

J Manipulation des nombres

Commençons à écrire quelques programmes qui utilisent les types *int* et *float*. Nous avons vu qu'en C, une variable doit être **déclarée**, tout comme son type. Par exemple, pour créer une variable x entière valant 5, on écrit:

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

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

```
1 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 11

Lisez le code suivant:

```
1 #include <stdio.h>
2
3 int main(){
4     int n = 15 ;
5     float prix = 6.99 ;
6
7     printf("J'ai acheté %d t-shirts à %f euros\n", n, prix);
8
9     return 0;
10 }
```

- Q1. Qu'afficherait le programme une fois compilé et exécuté?
- Q2. Créez un nouveau fichier C, recopiez le programme ci-dessus, compilez et exécutez. Le résultat est-il exactement 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 12

- Q1. Créez un nouveau fichier C.
- Q2. Créez deux variables flottantes $\boxed{x}$ et $\boxed{y}$ valant respectivement 5 et 2.
- Q3. Créez une nouvelle variable flottante $\boxed{z}$ contenant $\frac{x}{y}$ (en C, la division se fait grâce au symbole $\boxed{/}$). Affichez la valeur de cette variable.
- Q4. Compilez et exécutez le programme, et vérifiez qu'il affiche bien 2.5.
- Q5. Recommencez en utilisant des variables entières à la place. Que remarquez vous ?

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

Exercice 13

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

- Q1. Déclarez une variable `x` entière, valant 3
- Q2. Déclarez une variable `y` flottante, valant 0.25
- Q3. Déclarez une variable `somme_f` flottante valant $x + y$
- Q4. Déclarez une variable `somme_i` entière valant $x + y$
- Q5. 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 14

- Q1. Lisez le code suivant:

```
1  #include <stdio.h>
2
3  int main(){
4      int x, y; // permet de déclarer deux variables en même temps
5
6      printf("Entrez x: ");
7      scanf("%d", &x); // Attention: avec scanf, on précède
8                        // les variables du symbole '&'
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 à votre avis ?

- Q2. Recopiez ce code dans un nouveau fichier C et exécutez le. Testez avec plusieurs valeurs de `x` et `y`.
- Q3. 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 même avoir des comportements différents d'une exécution à l'autre !

Exercice 15

Q1. 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  }
```

Q2. Modifier le programme précédent pour qu'il fasse la multiplication de deux flottants plutôt que la somme de trois entiers.

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 16

Lisez le code suivant:

```
1  #include <stdio.h>
2
3  int main()
4  {
5      int x;
6      int y;
7
8      printf("Entrez x: ");
9      scanf("%d", &x);
10     printf("Entrez y: ");
11     scanf("%d", &y);
12
13     if (x < y){
14         printf("oui\n");
15     } else {
16         printf("non\n");
17     }
18
19     return 0;
20 }
```

En vous appuyant sur vos connaissances en anglais, prédisez ce qu'affiche ce programme. Recopiez le dans un nouveau fichier C afin de vérifier.

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

```
1  if (CONDITION){
2      /*
3      CODE SI VRAI
4      */
5  } else {
6      /*
7      CODE SI FAUX
8      */
9  }
```

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:

```
1  if (CONDITION){
2      /*
3      CODE
4      */
5  }
```

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 17

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

Pensez à bien tester vos programmes, en particulier pour le deuxième regardez ce qui se passe lorsque l'on rentre l'une des deux valeurs critiques.

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

- Le “ou”, noté `||`. Par exemple: `x == 0 || x == 3` est vérifiée lorsque 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` est vérifiée lorsque les deux conditions sont vérifiées, donc si x est strictement compris entre 0 et 3.
- Le “non”, noté `!`. Par exemple: `!(0 < x && x < 3)` est vérifiée lorsque la condition intérieure ne l'est pas, soit quand x n'est pas strictement contenue entre 0 et 3.

Exercice 18

Écrivez les programmes suivants. Essayez de minimiser le nombre de fois où vous utilisez `if`, en utilisant les opérateurs logiques à bon escient:

1. Un programme demandant de rentrer trois nombres et qui affiche celui qui est entre les deux autres (i.e. la médiane). Vérifiez que votre programme fonctionne toujours si plusieurs valeurs sont identiques
2. Un programme demandant de rentrer un entier n , et affichant:
 - “ga” si n est multiple de 3 mais pas de 5
 - “bu” si n est multiple de 5 mais pas de 3
 - “gabu” si n est multiple de 3 et de 5
 - n si n n'est multiple ni de 3 ni de 5

Vérifiez que dans tous les cas le retour à la ligne s'effectue comme il faut. **Aide:** `a % b` permet d'obtenir le reste modulo b de a .

Exercice 19

On souhaite modifier le programme de la question 2 de l'exercice 18 pour qu'il lise un entier n et réalise différents affichages selon les divisibilités de n :

Divisible par 3 ?	Divisible par 5 ?	Divisible par 7 ?	Affichage
X	X	X	n
✓	X	X	”ga”
X	✓	X	”bu”
✓	✓	X	”gabu”
X	X	✓	”zo”
✓	X	✓	”gazo”
X	✓	✓	”buzo”
✓	✓	✓	”gabuzo”

- Q1.** Déterminer la logique sous-jacente, et l'implémenter dans un programme le plus court possible, i.e. sans avoir 8 “si-sinon” testant les huit lignes de la table.

- Q2.** Créer une nouvelle version du programme, dans laquelle on teste aussi la divisibilité par 11, en l'associant à la syllabe "meu", en suivant la logique établie précédemment. La taille de votre programme ne doit pas doubler !

M Gestion du temps et de l'aléatoire

Exercice 20

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

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

(ne vous préoccupez pas de ce que veut dire NULL pour le moment.)

- Q1.** On rappelle que le type `int` peut stocker des entiers allant jusqu'à $2^{31} - 1$. En quelle année environ le résultat de la fonction `time` dépasserait-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. 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);
```

- Q2.** Jusqu'à quelle date environ peut-on utiliser ce format ?

- Q3.** 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 21

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 nombres en lui fournissant une *graine* / *seed*. Une bonne solution³ est de choisir comme seed la valeur renvoyée par la fonction `time` car elle change souvent (une fois par seconde).

Q1. 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 ?

Q2. Modifiez le programme pour donner comme seed à `srand` une valeur fixée de votre choix. Recompilez et exécutez le programme plusieurs fois: que constatez-vous ?

Q3. Enlevez totalement l'instruction `srand` et regardez comment le programme se comporte à présent.

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

Q4. Écrivez un programme qui tire et affiche un nombre aléatoire entre 0 et 99 inclus.

Q5. Écrivez un programme qui tire et affiche un nombre aléatoire entre 10 et 20 inclus.

Q6. Déterminez la formule générale pour générer un nombre aléatoire entre deux nombres a et b inclus (avec $a \leq b$).

A retenir L'instruction `srand(time(NULL));` s'utilise une seule fois au début de la fonction `main`, et sert à initialiser le générateur d'aléatoire. Dans tout le reste du programme, écrire `rand()%a` permet d'obtenir un nombre aléatoire entre 0 et $a - 1$ inclus.

³C'est une bonne solution dans le cadre du cours pour apprendre à coder en C, mais dans un vrai programme ça serait très dangereux car des utilisateurs malveillants pourraient "prédire l'aléatoire" !

N Exercice libre

Exercice 22

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 jeu demandant de résoudre une grosse addition, et qui affiche le temps mis pour la résoudre.
- Un programme calculant les solutions d'une équation du second degré
- Un quiz chronométré sur les trois œuvres du programme de français.
- 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.

Si vous souhaitez calculer des racines, des cosinus, ou d'autres opérations mathématiques classiques, vous aurez besoin de la librairie `<math.h>`. Pour l'utiliser, il faut non seulement écrire `#include <math.h>` au début de votre programme C, mais aussi rajouter l'option `-lm` (comme **Link Math**) à la fin de la commande de compilation:

```
gcc mon_programme.c -o nom_executable -lm
```

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