

TP1: Utilisation du terminal et premiers programmes en C

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

5 septembre 2022

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 utilisant très peu la souris. C'est une manière assez différente d'utiliser un ordinateur par rapport à ce dont vous avez 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


Pour installer la machine virtuelle avec le même environnement que sur les ordinateurs du lycée, suivez les instructions dans le dossier que vous avez récupéré à mon bureau.

Sur les ordi du lycée, pour lancer la machine virtuelle, ouvrez VMWare Workstation Player et démarrez la machine virtuelle des MP2I. Dans la machine virtuelle, au lancement, sélectionnez "Debian/Linux", puis connectez vous à votre session avec vos identifiants personnels.

ATTENTION Envoyez-vous votre travail par mail à la fin des sessions pour pouvoir continuer à travailler plus tard. En effet, vos sessions sur les ordinateurs du lycée ne sont *pas* partagées entre les ordi, donc vous devrez retourner sur la même machine physique pour retrouver votre travail si vous ne vous l'envoyez pas par mail ou par un autre moyen. Ceci est valable pour vos sessions Windows ET Debian.

A 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.

Exercice 1. Ouvrez un terminal en cliquant sur l'icône  dans la barre des raccourcis en haut de votre écran. Vous pouvez rajouter un raccourci clavier pour ouvrir un terminal (par exemple alt+t) en sélectionnant *Système > Préférences > Matériel > Raccourcis clavier*.

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... 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_mp2i*, qui est le dossier où se situeront tous vos fichiers personnels.

Il est tout de même possible de se déplacer dans l'ordinateur avec un explorateur de fichiers classique. L'explorateur de fichiers installé sur les machines Debian s'appelle **Caja** (vous demanderez aux LV2 Espagnol).

Il est aussi possible d'ouvrir un terminal directement dans un répertoire en y allant avec l'interface graphique classique, et en faisant clic-droit puis "ouvrir un terminal". Voyons quelques commandes utiles :

- 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 2.

1. Créez vous un répertoire pour les TP, puis accédez-y.
2. Créez à l'intérieur de ce répertoire un sous-répertoire pour le TP1, accédez y.
3. Créez un fichier texte appelé **blabla.txt**.
4. Affichez le contenu du dossier avec **ls**
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** ?

B 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 3.

1. Ouvrez un terminal, et places vous dans votre répertoire de TP, là où vous avez créé le dossier **TP1**. 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 "ajouter à l'archive" sur le dossier que vous voulez archiver.

C Rendu de TP

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

- Chaque TP doit être rendu au plus tard le dimanche suivant à 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, ainsi qu'un fichier texte appelé "réponses.txt" avec les réponses pour les questions nécessitant de rédiger quelque chose.
- 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 48 dans la classe, ça représente beaucoup de travail de vous corriger !

Tentez dans la mesure du possible de ne pas vous y mettre le dimanche à 21h et de le rendre un peu en avance. En effet :

Théorème 1. Le plus tôt vous rendez un TP/DM, le plus tôt je peux le corriger et vous le rendre.

De plus la page de dépôt de fichier se désactive à 22h00 exactement, prévoyez un peu d'avance.

D Édition de texte

Pour écrire du code, on utilise un éditeur de texte. Sur vos machines, vous avez 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 :

- **Pluma**
- **Visual Studio Code**
- **Sublime Text**

Ces éditeurs sont adaptés au code car ils colorent d'eux même les mots-clés des différents langages. De plus, Visual Studio Code et Sublime Text ont un système d'auto-complétion pour coder plus vite. Je vous conseille fortement d'utiliser Sublime Text mais ce n'est pas obligé ! Pour ouvrir un fichier dans Sublime Text, vous pouvez utiliser dans le terminal la commande **subl nom_du_fichier**.

E 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. Cela signifie également que l'on utilise 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 = x*x + 5*x - 4*x;
```

F Un langage typé

Le C est un langage typé, ce qui signifie que toutes les variables et les objets 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.

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 à la variable. Ainsi, les deux codes suivants se comportent pareil :

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

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

Vous remarquez 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 ayant une certaine spécification en utilisant le moins de caractères possible. Le cours d'informatique n'en est pas une.

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** :

- Sur les **float** :

```
1 float x = 5;  
2 float y = 2;  
3 float z = x/y;
```

à la fin de l'opération, z contient la valeur 2.5.

— Sur les *int* :

```
1 int x = 5;
2 int y = 2;
3 float z = x/y;
```

à la fin de l'opération, z contient la valeur 2.0 car c'est le quotient de la division euclidienne, même si l'on a précisé que z était un flottant.

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**. Tous les programmes que l'on utilise dans la vraie vie (Chrome, PowerPoint, Fortnite, Spotify, etc...) sont des exécutables !

```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("Bonjour\n");
6     return 0;
7 }
```

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 de mon_programme.c

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 mon_programme.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 de mon_programme.c en affiche_bonjour

H Premier programme C

Dans un programme C, il y a plusieurs choses à écrire avant même d'écrire le programme 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 programmes 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 ”standard in/out” (soit entrée/sortie standard). Pour déclarer que l’on utilise cette librairie, on commence notre programme par :

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

La fonction main Dans un programme C, on peut écrire plusieurs fonctions, c’est à dire des morceaux de code ayant des buts bien précis. La fonction **main** (“principale” en anglais) est la fonction 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.

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 :

```
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.

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 parle de **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 TP comment utiliser les commentaires à bon escient.

C’est parti On va enfin pouvoir écrire un programme. La tradition veut que lorsque l’on apprend un nouveau langage, le premier programme que l’on écrit affiche “Hello world”, mais vous pouvez afficher ce que vous voulez.

Exercice 4.

1. Ouvrez un éditeur de texte, et tapez :

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

Enregistrez le code dans un fichier, appelez le comme vous voulez mais l’extension doit être **.c** nécessairement.

2. Compilez votre code, en nommant l'exécutable *hello_world*.
3. Lancez l'exécutable en tapant `./hello_world`. Que remarquez-vous ? L'affichage est-il bien fait ?

Pour corriger le problème d'affichage, on rajoute un retour à la ligne. En C, et dans la plupart des langages, le retour à la ligne s'écrit “*\n*” (comme *newline*), et s'insère directement dans le texte à afficher.

4. Modifiez votre code pour rajouter un retour à la ligne et relancez votre programme. N'oubliez pas de recompiler avant !
5. Rajoutez un commentaire sur une seule ligne expliquant ce que fait votre code puis un commentaire multi-lignes avec votre prénom, nom, et la date d'aujourd'hui.

I 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 :

```
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 ce que l'on affiche que l'on souhaite insérer un entier ou un flottant, respectivement, puis on spécifie les nombres à afficher, dans le bon ordre.

Exercice 5. 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 }
```

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 ?

Une fois qu'une variable est déclarée, on peut l'utiliser comme on veut, et en particulier lui assigner une nouvelle valeur sans la redéclarer à chaque fois. Sur les entiers et les flottants, on peut utiliser les opérations mathématiques classiques (+, −, ×, /) comme sur une calculatrice. Le symbole pour la multiplication est l'astérisque *.

Exercice 6. Lisez le programme suivant :

```

1 #include <stdio.h>
2
3 int main(){
4     int x = 5;
5     printf("première valeur de x: %d\n", x);
6
7     x = 2 * x + 3;
8     printf("deuxième valeur de x: %d\n", x);
9
10    return 0;
11 }

```

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 ?

Exercice 7. Essayons de mélanger les types. Créez un nouveau fichier C.

1. Déclarez une variable x entière, valant 3
2. Déclarez une variable y flottante, valant 0.25
3. Déclarez une variable somme_f flottante valant $x + y$
4. Déclarez une variable somme_i entière valant $x + y$
5. Affichez les 4 variables. Que remarquez vous ?

Avant de continuer, nous allons apprendre à rendre nos programmes interactifs, c'est à dire leur permettre de lire ce que l'on écrit dans le terminal. On utilise la fonction ***scanf*** qui scanne le terminal et lit ce que l'on y écrit, jusqu'à rencontrer un retour à la ligne. Elle s'utilise de manière similaire à `printf` en précisant avec `%d` et `%f` ce qu'il faut scanner.

Exercice 8.

1. 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
10    scanf("%d", &x);
11
12    printf("Entrez y: ");
13
14    scanf("%d", &y);
15
16    printf("x vaut %d, y vaut %d, la somme vaut %d\n",
17          x, y, x+y);
18
19    return 0;
20 }

```

Que fait-il ?

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

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 !

J 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 9.

1. Regardez 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 }
```

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

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

```
1  if (CONDITION){
2      /*
3      CODE
4      */
5  } else {
6      /*
7      CODE
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 $x \leq y$
- Le test de supériorité stricte (" $>$ ") s'écrit $x > y$
- Le test de supériorité large (" \geq ") s'écrit $x \geq y$

Exercice 10. Écrivez les programmes suivants 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 : $x > 0 \ \&\& \ x < 3$ sera vérifiée si, et seulement si, les deux conditions sont vérifiées, donc si x vaut entre 0 et 3.
- Le "non", noté `!`. Par exemple : $!(x == 0)$ sera vérifiée si, et seulement si, la condition intérieure ne l'est pas, donc si x n'est pas nul.

Exercice 11. 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 .

K Gestion du temps et de l'aléatoire

Exercice 12. 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, on verra ça plus tard.

1. Ce nombre peut-il tenir dans une variable de type `int` ? En quelle année environ dépasserait-il la limite sur un `int` ?

La fonction `time` ne renvoie en fait pas un `int`, mais un ***long int***, qui fait 64 bits sur les machines 64-bits, ce qui est le cas des machines du lycée. Il faut donc stocker son 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);
```

2. Jusqu'à quelle date peut on utiliser ce format ? ***Attention*** : comme pour le type `int`, le type `long int` stocke des entiers positifs ou négatifs.
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 13. 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 pour l'instant est de choisir comme *seed* la valeur renvoyée par la fonction `time`.

1. Recopiez, et compilez le code suivant :

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

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

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)

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

3. Écrivez un programme qui tire et affiche un nombre aléatoire entre 0 et 99 ;
4. Écrivez un programme qui tire et affiche trois nombres aléatoires entre 10 et 20 ;

L Exercice libre

Exercice 14. 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. 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 (Vous pouvez vous contenter des cas où les solutions sont réelles).
- Un jeu demandant de résoudre une grosse addition, et qui affiche le temps mis pour la résoudre.