1 Utilisation de `git`

`git` est un outil permettant de synchroniser le code d'un projet entre plusieurs machines (et éventuellement plusieurs utilisateurs). Les informations à synchroniser sont en général centralisées sur un serveur dédié, et les utilisateurs peuvent soit demander au serveur de récupérer la dernière version du code source du projet, ou lui envoyer une nouvelle version du code source.

L'un des serveurs `git` le plus populaire est `gitlab`. Pour le début de ce TP, nous allons apprendre les bases de l'utilisation de `git`.

Vous pourrez vous en servir, par exemple, pour :

- sauvegarder votre travail en TP, et le récupérer chez vous ;
- sauvegarder un code que vous avez fait chez vous, et le récupérer sur l'ordinateur du TP ;
- collaborer sur un projet à plusieurs.

Exercice 1 (Création d'un projet git) :

1. Aller sur le site `https://gitlab.com/users/sign_up`, et créer un compte si vous n'en n'avez pas déjà un.
2. Une fois votre compte créé, créer un nouveau projet personnel, que vous appellerez "`mp2i`".
3. Sur la page de votre projet, cliquer sur "clone" (bouton bleu), et copier l'adresse "**Clone with HTTPS**" qui apparait.
4. Sur votre ordinateur, ouvrir une console, créer un dossier portant votre nom (avec `mkdir`).

La commande `git clone` permet de cloner un projet `git`, c'est-à-dire d'importer un projet git existant.

5. Aller dans le dossier (avec `cd`), et taper la commande : `git clone <adresse_github_copiée>`
6. Si tout va bien, un dossier "`mp2i`" vient d'apparaître (vérifier avec `ls`).

Exercice 2 (Utilisation de git) : Une fois dans un dossier synchronisé par `git`, vous pouvez interagir avec le serveur Gitlab à l'aide de commandes du programme `git` (à chaque fois, Gitlab vous demandera votre identifiant et votre mot de passe [1]).

1. La commande `git pull` permet de récupérer la dernière version du code présente sur le serveur. Tester cette commande.
2. Créer (et sauvegarder) un fichier "`hello.c`" contenant le code suivant :

```c
#include <stdio.h>

int main()
{
  printf("Hello World!\n");
}
```

3. La commande `git status` permet de voir si des fichiers ne sont pas suivis par `git`, ou si des fichiers suivis ont été modifiés. Tester cette commande.
4. La commande `git add <fichier>` permet d'ajouter un fichier à la liste de ceux suivis par `git`. On peut ajouter plusieurs fichiers d'un coup avec `git add fichier1 fichier2 ... fichierN`.
   Taper la commande : `git add hello.c`
5. Observer ce qui a changé avec `git status`.
6. La commande `git commit` permet de soumettre les dernières modifications d'un ou plusieurs fichiers à `git`. Chaque `commit` doit être accompagné d'un message décrivant ce `commit`, que l'on peut donner avec l'option `-m`. Un commit se fera donc via une instruction de la forme :
   `git commit fichier1 fichier2 ... fichierN -m "Message du commit"`.
   Taper la commande : `git commit hello.c -m "Ajout de hello.c"`

1. Il est possible de régler Gitlab pour qu'il se souvienne de votre ordinateur (il faut alors le cloner avec "**Clone with SSH**" et paramétrer la clé SSH). Je vous conseille d'utiliser une clé SSH sur votre ordinateur personnel (ça sera plus pratique), mais de ne pas le faire sur les ordinateurs de Janson (car quelqu'un d'autre pourrait modifier votre répertoire gitlab sans connaître votre mot de passe).

7. Observer ce qui a changé avec `git status`.

Normalement, `git` vous indique que vous avez désormais un `commit` d'avance par rapport au serveur. Cela signifie que git a bien enregistré vos modifications sur votre ordinateur, mais n'a pas encore envoyé ces informations au serveur Gitlab.

8. La commande `git push` permet d'envoyer tous les commits effectués au serveur.
   Tester cette commande.
9. Observer ce qui a changé avec `git status`.

Il existe pleins d'autres commandes `git` plus avancées (que l'on ne va pas détailler ici). On peut par exemple demander l'historique des modifications, et revenir à une ancienne version de notre projet.

De plus, pour un projet à plusieurs, l'intérêt d'utiliser git est qu'on va pouvoir facilement gérer les confits (plusieurs utilisateurs qui travaillent en parallèle sur le projet).

Une dernière astuce : si vous voulez que `git status` ignore certains fichiers que vous ne voulez pas synchroniser (par exemple, les fichiers créés suite à la compilation de votre code), vous pouvez créer un fichier `.gitignore` contenant les fichiers à ignorer (un par ligne).

2 Tableaux et pointeurs

Exercice 3 : On souhaite écrire une fonction `echange` qui échange la valeur de deux variables entières.

```
1 void echange(int a, int b)
2 {
3   int tmp = a;
4   a = b;
5   b = tmp;
6 }
```

1. Le code ci-dessus ne fonctionne pas. Pourquoi ?
2. proposer une fonction `echange` qui permette d'échanger deux variables entières.
3. Écrire une fonction `void affiche_entiers(int *tab, int n)` qui affiche les valeurs d'un tableau d'entiers de taille n.
4. Écrire une fonction `void affiche_floats(float *tab, int n)` qui affiche les valeurs d'un tableau de flottants de taille n.

Exercice 4 : On rappelle qu'on peut allouer nous-même de la mémoire à l'aide de la fonction `malloc`, et de la libérer avec la fonction `free`.

1. Écrire une fonction `int *copie(int *tab, int n)` qui prend en arguments un tableau d'entiers `tab` et sa taille n, et qui renvoie une copie de te tableau.
2. Tester que votre code fournit bien une copie : dans la fonction `main`, créer un tableau `tab1` et sa copie `tab2`, puis modifier `tab2` et afficher les valeurs des deux tableaux à l'aide de la fonction `affiche_entiers`.

Exercice 5 : On considère une suite numérique $(u_n)_{n\in\mathbb{N}}$ donéne par son premier terme u_0 et la relation de récurrence $u_{n+1} = a \times u_n + b$, où a et b sont des réels.

1. Écrire une fonction `float *premiers_termes(float u0, float a, float b, int n)` qui renvoie un tableau de taille n contenant les n premiers termes de la suite.
2. Tester votre programme à l'aide de la fonction `affiche_floats`.

Exercice 6 (Crible d'Ératosthène) : Le crible d'Ératosthène est un algorithme permettant de trouver efficacement les nombres premiers inférieurs à n. Son principe est le suivant :

- On crée un tableau `tab` de booléens de taille $n + 1$. Le but est que `tab[i]` contienne `true` ssi i est premier.

- On initialise `tab[0]` et `tab[1]` à `false`, et toutes les autres cases à `true`.
- On parcourt toutes les cases d'indice $i \in [\![2, n]\!]$, et :
  - Si `tab[i]` vaut `true`, on met `false` dans toutes les cases d'indice $j = k \times i$, avec $k \geq 2$ et $j \leq n$.
  - Sinon, on ne fait rien.
- On renvoie le tableau.

1. Écrire une fonction `bool *eratosthene(int n)` implémentant cet algorithme.
2. Tester votre programme.

Exercice 7 (Triangle de Pascal) : Dans cet exercice, on veut créer des tableaux `t` de taille $n+1$ tels que `t[i]` $= \binom{n}{i}$ (c'est la n-ème ligne du triangle de Pascal).

Exemple

```
int t0[] = {1};
int t1[] = {1,1};
int t2[] = {1,2,1};
int t3[] = {1,3,3,1};
```

Étant donné un tel tableau `t` (pour un certain n), on peut construire le tableau représentant la ligne $n+1$ comme suit :

- on crée un tableau de taille $n+2$;
- on met 1 dans la première et la dernière case ;
- pour tout autre case (d'indice i), on met la valeur de `t[i-1] + t[i]`.

1. Écrire une fonction `int *prochaine_ligne(int *t, int n)` implémentant cet algorithme.
2. Écrire une fonction `int **triangle_de_pascal(int n)` qui renvoie un tableau contenant les n premières lignes du triangle de Pascal.
3. Afficher les 10 premières lignes du triangle de Pascal.

Exercice 8 (Circulation automobile) : Dans cet exercice nous allons simuler (de façon très simplifiée) la circulation automobile sur une route à une seule voie (et dans un seul sens, de droite à gauche). L'idée est la suivante : on discrétise la route qui est donc assimilée à un tableau de petits entiers à n cases (0 : la case est libre, sinon elle est occupée) ; si au temps t, un véhicule v se trouve en case i, alors :

- si la case $i-1$ est libre au temps t, le véhicule v se retrouvera en case $i-1$ au temps $t+1$;
- si la case $i-1$ est occupée au temps t, le véhicule v reste en case i au temps $t+1$.

Pour les cases extrémales, on impose le comportement suivant :

- le véhicule en position 0 au temps t disparaît au temps $t+1$;
- si la case $n-1$ est libre au temps t, alors un véhicule arrive de façon aléatoire au temps $t+1$;
- si la case $n-1$ est occupée au temps t, aucun véhicule n'arrive dans cette case au temps $t+1$.

Nous allons fabriquer un modèle où il y a 4 types de véhicules. Un case de la route contiendra donc une valeur dans $[\![0, 4]\!]$, 0 désignant une case libre, et les autres valeurs représentant ces 4 types de véhicules.

Remarque : vous pouvez trouver sur ma page web le fichier `voiture.c` qui contient les fonctions de la page suivante.

La fonction suivante permet de renvoyer 0 avec probabilité $1/f$ (f est la fréquence d'apparition), et un entier représentant un véhicule sinon (chaque véhicule avec un probabilité 1/4) :

```c
#include <time.h>

srand(time(0)); // Initialisation de la seed

int vehicule(int f)
{
  if (rand()%f == 0)
  {
      return 0;
  }
  else
  {
      return (rand()%4) + 1;
  }
}
```

1. Écrire une fonction `int *init(int f, int n)` qui crée un tableau de taille n et initialise toutes ses cases à l'aide de la fonction `vehicule`, puis renvoie ce tableau.
2. Écrire une fonction `int *next(int *route, int n, int f)` qui prend en arguments un tableau de taille n représentant l'état de la route, et renvoie un tableau représentant le prochain état de la route (f est la fréquence d'apparition).

La fonction suivante permet d'afficher la route à un moment donné :

```c
#include <stdio.h>
#include <wchar.h>
#include <unistd.h>

void affiche(int *route, int n){
  //le type wint_t est hors programme
  wint_t v[] = {' ',(wint_t)128663, (wint_t)128665, (wint_t)128661, (wint_t)127949};
  //cette instruction est hors programme et permet l'affichage des emojis (voitures)
  setlocale(LC_ALL, "");
  for(int i=0; i<n; i++){
    printf("%lc", v[route[i]]);
    if (route[i] == 0)
    {
      printf("%lc", v[route[i]]);
    }
  }
  printf(" ");
  // instructions également hors programme
  fflush(stdout); // force à mettre à jour la sortie standard
  sleep(1); // Met le programme en pause pendant 1 seconde
  printf("\r"); // revient au début de la ligne
}
```

3. Dans la fonction `main` de votre programme, initialiser une route de taille 60 avec $f = 2$, et afficher l'état de la route du temps 0 jusqu'au temps 30 (à tester dans le terminal intégré à VSCode pour un meilleur rendu).
4. Votre programme a-t-il des fuites de mémoire (des tableaux créés par `malloc` mais non libérés par `free`) ? Si oui, corriger ces fuites de mémoire.