TD Machine #1 de CS101

- Durée 1h30
- Pas de compte-rendu. Il est fortement recommandé de prendre des notes personnelles.
- Version 2022.1
- *author* Laure Gonnord

À vos marques

- Le TDM est réalisé sur les machines de TP Linux de la salle B141.
- Essayer le plus possible de faire seul·e (on apprend moins vite en binôme)
- Ne pas hésiter à demander de l'aide (on apprend!)
- Créer un répertoire pour les TPs de C nommé CS101, et travailler dans un sous répertoire nommé TDM1.

Prêts - Hello World

On réalise tous ensemble notre premier programme C (éventuellement votre chargé·e de TP peut réaliser une démo à l'aide du vidéo projecteur).

- Dans un éditeur (qu'on laisse ouvert!), recopier le programme C suivant:

```
int main(){
    printf("Hello world\n!")
    return 0;
}
```

- Sauvegarder (ne pas fermer l'éditeur): lui donner le nom `hello.c`.
- Ouvrir un terminal *au même endroit que le fichier*, et compiler: avec la commande `clang hello.c -Wall -Werror -o hello`
- Corriger les erreurs (cf les slides du cours), jusqu'à faire taire le compilateur. Laisser ce terminal ouvert.
- Exécuter avec la commande `./hello` toujours dans le même terminal.

Partez : premiers programmes simples

Exo 1: min et max

- Dans votre éditeur, trouver comment créer un nouveau fichier .c nommé `minmax.c`, toujours dans le même répertoire `CS101/TDM1`.
- Créer un programme C "minimal" contenant les lignes suivantes *merci de ne pas copier-coller!*:

```
#include <stdio.h>
int main(){


    return 0;
}
```

- Vérifier qu'il compile: `clang minmax.c -Wall -Werror -o minmax`. *On réalisera toujours une manipulation de ce style au début de chaque exercice*.
- Déclarer un entier nommé `x`, initialisé à 42. De même déclarer un entier `y`, initialisé à 24.
- Vérifier l'initialisation en utilisant une impression: `printf(.....)` (avec la macro d'affichage `%d`.) Compiler, tester.
- Utiliser la conditionnelle pour déterminer qui de x ou de y est le min, et le max, et réaliser un affichage qui ressemble à:

```
le minimum est : 24, et le maximum est 42.
```

- Compiler, tester. Modifier les valeurs de x et y, compiler, tester.

Exo 2: devine nombre

- Dans un nouveau fichier nommé `devine.c`, écrire les lignes suivantes (toujours sans copier coller merci)

```
#include <stdio.h>
int main(){

    int r = random() %100 +1 ;

    printf("%d\n", r);


    return 0;
}
```

(oui, c'est (presque) pareil qu'avant, mais la répétition est importante!)

Ce programme utilise la fonction random déclarée dans le fichier d'entête `stdlib.h`, cette fonction retourne un entier quelconque, donc on utilise le modulo (`%`) pour récupérer un entier entre 0 et 99.

- Compiler, remarquer le warning, et ajouter dans l'entête `#include <stdlib.h>`
- Compiler, exécuter, et remarquer que le nombre est toujours identique. Pour résoudre ce problème, on peut lire la documentation de random (`man 3 random` dans le terminal), et ajouter `srandom(time(NULL));` (il faudra ajouter le fichier d'entete `time.h`). Verifier que tout fonctionne. À ce stade la variable r contient une valeur aléatoire entre 1 et 100 compris.
- On rappelle que `scanf("%d",&y)` permet de récupérer une valeur entière entrée au clavier par l'utilisateur·rice dans la variable y supposée déclarée. Demander une valeur à l'utilisateur, et faire afficher "trouvé", si cette valeur est égale à r. Tester (on peut pour l'instant laisser l'impression de r, cela permet de vérifier". Compiler, tester.
- Modifier le programme pour réaliser un "devine nombre" avec un nombre de coups égal à 10 au maximum.

Régime de croisière : des fonctions.

- Dans un fichier nommé `fonctions.c` nous allons réaliser une série de petites fonctions classiques sur des nombres entiers. Une partie de ces fonctions a déjà été écrite en cours, on vous demande d'essayer de vous y reporter le moins possible.

Préparation

```
#include <stdio.h>

int somme(int n){
  // code à remplir

  return 1515;
}

int main(){
  int x=5;
  int resu = somme(x);
  printf("la somme 1+...+ %d est %d\n ", x, resu);

  return 0;
}
```

- On prépare le programme, on compile, on corrige les erreurs de compilation, et on exécute.

Somme

- La fonction somme retourne 1515, ce qui n'est pas ce qu'on veut. Modifier cette fonction pour qu'elle calcule et retourne la somme des entiers de 1 à n, n étant passé en paramètre. Compiler, corriger, tester, etc.

Factorielle

- De la même façon, dans le même fichier, écrire une fonction qui calcule le produit des entiers de 1 à n. On utilisera la même valeur de x pour appeler cette fonction dans le *main*. Le nom de cette fonction est à décider.
- Compiler, corriger, tester.

Puissance

- Écrire une fonction prenant deux paramètres entiers a et n et calculant a^n (en utilisant n-1 multiplications).

Loterie

- Effectuer une suite de tirages aléatoires (de 3 nombres inférieurs à 1000) jusqu'à obtenir un triplet composé d'un nombre pair suivi de deux nombres impairs. La fonction `loterie` effectuera les impressions, et retournera le nombre de tirages effectués. (On rappelle que modulo est `%` en C) Exemple d'exécution:

```
Loterie
874, 34, 869
915, 444, 462
6, 776, 367
66, 425, 85
Résultat obtenu en 4 coups
```

Calcul de nième terme d'une suite.

On définit la suite récurrente u(0)=1, u(n+1)=1+2*u(n).

- Écrire une fonction (itérative) qui calcule le n-ième terme de la suite.
- Écrire une fonction qui retourne le premier n tel que `u(n)>1000`.
- On continue à bien tester au fur et à mesure.

Suite récurrente double

On définit la suite récurrente u(0)=1, u(1)=1, u(n+2)=u(n+1)+u(n).

- Écrire une fonction (itérative) qui calcule le n-ième terme de la suite (de fibonacci)
- Lire la magnifique page https://fr.wikipedia.org/wiki/Suite_de_Fibonacci

PGCD (euclide) - itératif

- Lire la page Wikipédia de l'algorithme d'euclide pour le pgcd: https://fr.wikipedia.org/wiki/Algorithme_
  repérer l'algorithme itératif dans le texte, l'implémenter comme une fonction:

```c
int pgcd(int a, int b){
    // compléter - on suppose que a, b >0
}
```

- Tester, etc

Récursivité

- Écrire une version récursive des (3) fonctions précédentes.