1. Écrivez une fonction **max** qui prend deux entiers **a** et **b** en paramètres et renvoie le plus grand des deux. Écrivez également une fonction **main** qui appelle votre fonction.
2. Écrivez une fonction **est_espace** qui prend un caractère **c** en paramètre et renvoie si **c** est un caractère de type espace ou non.
   Disons que ces caractères sont ' ', '\t' et '\n' pour simplifier le problème.
   Écrivez également une fonction **main** qui appelle votre fonction.
3. En C, la fonction **rand** renvoie un nombre entier pseudo-aléatoire entre 0 et RAND_MAX (bornes incluses). Voici le prototype de cette fonction :

   ```c
   #include <stdlib.h>
   int rand(void);
   ```

   Les nombres renvoyés par **rand** sont rarement vraiment aléatoires. En général, un nouveau nombre est généré à partir du précédent grâce à un calcul.
   Afin d'obtenir des résultats différents à chaque lancer du programme, il faut initialiser la **graine aléatoire** (la valeur initiale du nombre aléatoire) au début du programme grâce à la fonction **srand**. Pour cela, on utilise en général le temps courant grâce à la fonction **time**.

   ```c
   #include <time.h>
   srand(time(0)); // à mettre au début du main
   ```

   Écrivez trois fonctions suivantes :

   ```c
   int jet_de_6(void); // Dé à 6 faces (1 -> 6)
   int jet_de_n(int n); // Dé à n faces (1 -> n)
   int jet_piece(void); // 0 pour pile, 1 pour face
   ```

   Faites un **main** appelant vos fonctions.
4. Soit n un entier. Soient f et g deux fonctions $\mathbb{N} \to \mathbb{N}$ définies par

$$f(n) = \begin{cases} 0 & \text{si } n \leq 0 \\ g(n-1) & \text{sinon} \end{cases}$$

$$g(n) = \begin{cases} 1 & \text{si } n \leq 0 \\ n + f(n/2) & \text{sinon} \end{cases}$$

   Implémentez ces deux fonctions en C.
   Ajoutez une fonction **main** qui affiche le résultat de de $g(3)$.

5. La suite de Fibonacci peut être définie comme une fonction $\mathbb{N} \to \mathbb{N}$:

$$f(n) = \begin{cases} 0 & \text{si } n = 0 \\ 1 & \text{si } n = 1 \\ f(n-1) + f(n-2) & \text{sinon} \end{cases}$$

   Écrivez une fonction récursive **fibbo_rec** qui calcule et renvoie **f(n)** de manière naïve : en appelant explicitement $fibbo\_rec(n-1)$ et $fibbo\_rec(n-2)$.

   Écrivez une fonction itérative **fibbo_iter** qui calcule et renvoie **f(n)**. Pour cela, stockez les deux derniers termes calculés dans des variables *temporaires*.

   Dessinez l'arbre d'appel de $fibbo\_rec(5)$. Comparez le nombre d'additions effectuées dans cet appel à celui fait lors d'un appel de $fibbo\_iter(5)$.
   Que risque-t-il de se passer lors d'un appel de $fibbo\_rec$ lorsque n devient grand ?

6. Que fait le code suivant ?

```c
#include <stdio.h>
int dat_function(int n, int k) {
    return dat_function(k, n);
}
int main() {
    dat_function(42, 37);
    return 0;
}
```

7. En compilant son fichier, Derp a obtenu le warning suivant :

```
err.c:3:5: warning: implicit declaration of function 'func'
           is invalid in C99 [-Wimplicit-function-declaration]
```

   Expliquez ce que veut dire ce warning. Quelles erreurs Derp a-t-il pu commettre ?

8. En compilant son fichier, Derp a obtenu l'erreur suivante :

```
$ clang99 err.c
/tmp/err-b2fc60.o : Dans la fonction « main » :
err.c:(.text+0x1a) : référence indéfinie vers « my_print »
clang-3.8: error: linker command failed with exit code 1
           (use -v to see invocation)
```

   Expliquez ce que veut dire cette erreur. D'où peut-elle venir ?

9. Que fait ce code ? Que calcule la fonction $f(i, j)$?

```c
#include <stdio.h>
int g(int i) { return i+1; }
int h() { return 1; }
int f(int i, int j) {
    int t = h(), u = h();
    while (t <= i) {
        u *= j;
        t += h();
    }
    return u; }
int main() {
    printf("f(2,6)=%d\n", f(2,6));
    printf("f(3,5)=%d\n", f(3,5));
    return 0; }
```