Programmation Impérative 2 - L2 Informatique

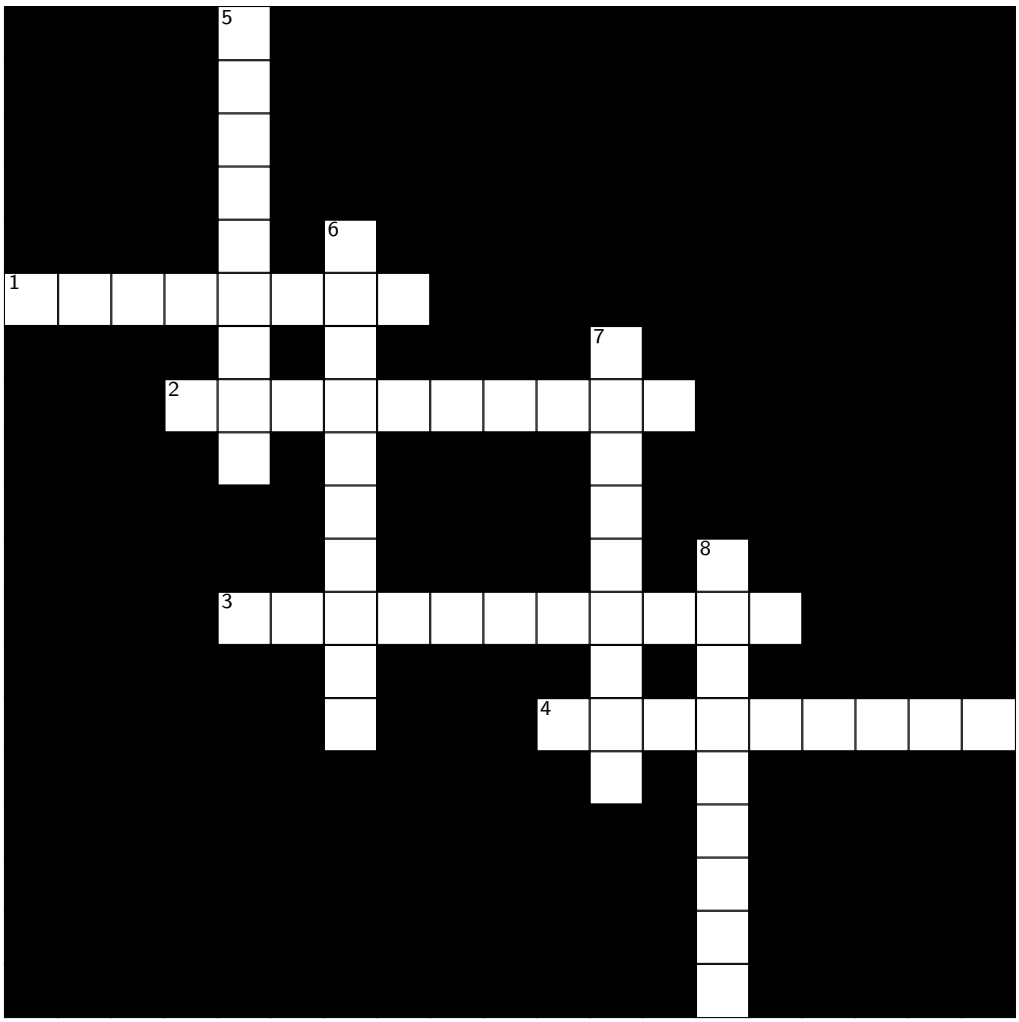
TD 2

Exercice I -

Ecrire une fonction récursive qui calcule x^n pour x un réel quelconque et n un entier positif ou nul (pensez à fixer le cas terminal). On suivra le principe d'algorithme suivant :

- si n est pair, alors $x^n = (x^2)^{n/2}$
- si n est impair alors $x^n = x * x^{n-1}$.

Exercice II - Essayez de remplir la grille de mots croisés à partir du code suivant. On supposera que la fonction `print_reverse` imprime une chaîne de caractères en ordre inverse.



```
int main() {
   char *juices[] = {"dragonfruit", "waterberry", "sharonfruit", "uglifruit", "rumberry",
           "kiwifruit", "mulberry", "strawberry", "blueberry", "blackberry", "starfruit"};
   char *a;

1  puts(juices[6]);
2  print_reverse(juices[7]);

   a = juices[2];
   juices[2] = juices[8];
   juices[8]=a;

3  puts(juices[8]);
4  print_reverse(juices[(18 + 7) / 5]);
5  puts(juices[2]);
6  print_reverse(juices[9]);

   juices[1] = juices[3];

7  puts(juices[10]);
8  print_reverse(juices[1]);

   return EXIT_SUCCESS;
}
```

Exercice III - Pas de questions stupides, reliez les questions au réponses (en anglais). L'ordre des questions (et des réponses) peut avoir une importance.

1. There's a stdout and a stderr. Is there a stdin?
2. Can I print to it?
3. Can I read from it?
4. So is `fscanf(stdin, ...)` exactly the same as `scanf(...)`?
5. Can I redirect the standard error?
6. So I could write `my_program 2> errors.txt` ?
7. Why is it important that small tools use the Standard Input and Standard Output?
8. Why does that matter?
9. What is a pipe, actually?
10. So if two programs are piped together, does the first program have to finish running before the second program can start?
11. Why do small tools use text?
12. Can I connect several programs together with pipes?
13. If several processes are connected together with pipes and then I use `>` and `<` to redirect the Standard Input and Output, which processes will have their input and output redirected?
14. If I run the following command line
    ```
    (./my_program1 | ./my_program2)
       < myinput.csv > myoutput.txt
    ```
    Are the parentheses really necessary?

a. Yes. The parentheses will make sure the data file is read by the Standard Input of the `my_program1` program.

b. Yes, `>` redirects the Standard Output. But `2>` redirects the Standard Error.

c. Yes, they are identical. In fact, behind the scenes, `scanf(...)` just calls `fscanf(stdin, ...)`.

d. Small tools usually don't solve an entire problem on their own, just a small technical problem, like converting data from one format to another. But if you can combine them together, then you can solve large problems.

e. The `<` will send a file's contents to the first process in the pipeline. The `>`. will capture the Standard Output from the last process in the pipeline.

f. No, the Standard Input cannot be printed to.

g. Yes, by using `fscanf()`, which is just like `scanf()`, but you can specify the data stream.

h. No, Both of the programs will run at the same time; as output is produced by the first program, it can be consumed by the second program.

i. Yes.

j. It's the most open format. If a small tool uses text, it means that any other programmer can easily read and understand the output just by using a text editor. Binary formats are normally obscure and hard to understand.

k. Yes, just add more `|` between each program name. A series of connected processes is called a pipeline.

l. Because it makes it easier to connect tools together with pipes.

m. The exact details depend on the operating system. Pipes might be made from sections of memory or temporary files. The important thing is that they accept data in one end, and send the data out of the other in sequence.

n. Yes and as you probably guessed, it refers to the Standard Input.

Exercice IV - Ecrire une fonction qui prend en argument une chaîne de caractères et qui l'affiche dans le sens inverse de l'ordre de lecture usuel. Donner une version itérative et une version récursive.

Exercice V - Complétez les trous.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
   char line[80];

   FILE *in = fopen("spooky.csv", ..... );
   FILE *file1 = fopen("ufos.csv", .....);
   FILE *file2 = fopen("disappearances.csv", .....);
   FILE *file3 = fopen("others.csv", .....);

   while (..... (in, "%79[^\n]\n", line) == 1) {
      if (strstr(line, "UFO"))
          ..... (file1, "%s\n", line);
      else if (strstr(line, "Disappearance"))
          ..... (file2, "%s\n", line);
      else
          ..... (file3, "%s\n", line);
   }

   .....(file1);
   .....(file2);
   .....(file3);

  return EXIT_SUCCESS;
}
```

Exercice VI - Supposons que le code à trous ci-dessous se lance avec la ligne de commande:
`./categorize mermaid mermaid.csv Elvis elvises.csv the_rest.csv`
complétez les trous avec les mots suivants: `6`, `5`, `argc`, `argv[2]`, `argv[4]`, `argv[5]`, `argv[1]`, `argv[3]`.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[])
{
   char line[80];

   if (.......... != ........... ) {
      fprintf(stderr, "You need to give 5 arguments\n");
      return EXIT_FAILURE;
   }

   FILE *in = fopen("spooky.csv", "r");
   FILE *file1 = fopen(............ , "w");
   FILE *file2 = fopen(............ , "w");
   FILE *file3 = fopen(............ , "w");


   while (fscanf(in, "%79[^\n]\n", line) == 1)
   {
       if (strstr(line, ................))
          fprintf(file1, "%s\n", line);
       else if (strstr(line, ................))
          fprintf(file2, "%s\n", line);
       else
          fprintf(file3, "%s\n", line);
   }

   fclose(file1);
   fclose(file2);
   fclose(file3);

   return EXIT_SUCCESS;
}
```

Exercice VII - Questions/Réponses

1. Why are data types different on different operating systems? Wouldn't it be less confusing to make them all the same?
2. In what way?
3. What do 8-bit and 64-bit actually mean?
4. So what does that have to do with the size of ints and doubles?
5. I understand how whole numbers like ints work, but how are floats and doubles stored? How does the computer represent a number with a decimal point?
6. Do I really need to understand how floating-point numbers work?

a. It's complicated. Most computers used a standard published by the IEEE (`http://ieeexplore.ieee.org/document/4610935/`.

b. Technically, the bit size of a computer can refer to several things, such as the size of its CPU instructions or the amount of data the CPU can read from memory. The bit size is really the favored size of numbers that the computer can deal with.

c. When C was first created, most machines were 8-bit. Now, most machines are 32- or 64-bit. Because C doesn't specify the exact size of its data types, it's been able to adapt over time. And as newer machines are created, C will be able to make the most of them as well.

d. No. The vast majority of developers use floats and doubles without worrying about the details.

e. If a computer is optimized best to work with 32-bit numbers, it makes sense if the basic data type - the `int` - is set at 32 bits.

f. C uses different data types on different operating systems and processors because that allows it to make the most out of the hardware.

Exercice VIII - Jouez au compilateur. Voici une partie de programme:

```
  printf("A day on Mercury is %f hours\n", day);
  return EXIT_SUCCESS;
}

float mercury_day_in_earth_days() {
  return 58.65;
}

int hours_in_an_earth_day() {
  return 24;
}
```

Indiquez pour les 4 bouts de code suivant s'ils peuvent se mettre avant la partie de programme précédente. Indiquez pour chacun d'eux si le programme entier peut compiler, s'il faut afficher un warning et si le programme va fonctionner.

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

   float mercury_day_in_earth_days();
   int hours_in_an_earth_day();

   int main() {
     float length_of_day = mercury_day_in_earth_days();
     int hours = hours_in_an_earth_day();
     float day = length_of_day * hours;
   ```

2. ```
   #include <stdio.h>
   #include <stdlib.h>


   float mercury_day_in_earth_days();

   int main() {
     float length_of_day = mercury_day_in_earth_days();
     int hours = hours_in_an_earth_day();
     float day = length_of_day * hours;
   ```

3. ```
   #include <stdio.h>
   #include <stdlib.h>

   int main() {
     float length_of_day = mercury_day_in_earth_days();
     int hours = hours_in_an_earth_day();
     float day = length_of_day * hours;
   ```

4. ```
   #include <stdio.h>
   #include <stdlib.h>

   float mercury_day_in_earth_days();
   int hours_in_an_earth_day();

   int main() {
     int length_of_day = mercury_day_in_earth_days();
     int hours = hours_in_an_earth_day();
     float day = length_of_day * hours;
   ```

Exercice IX -

Une chaîne de caractères est un palindrome si elle a 1 seul caractère, ou bien si son premier et son dernier caractère sont identiques et si en les retirant la sous-chaîne obtenue est également un palindrome. (exemple : ressasser). Ecrire une fonction renvoyant 1 si un mot est un palindrome et 0 sinon (donner une version itérative et une version récursive).

Exercice X - Remettez dans l'ordre les instructions suivantes:

1. `gcc -Wall oggswing.c -o oggswing`
2. `whitennerdy.ogg`
3. `oggswing whitennerdy.ogg swing.ogg`
4. `oggswing`
5. `[SPACES]`
6. `oggswing.c`
7. `oggswing.h`
8. `[SPACES]`
9. `[TAB]`
10. `[TAB]`
11. `oggswing:`
12. `swing.ogg:`

Exercice XI - Questions/réponses

1. So I don't need to have declarations for int functions?
2. I'm confused. You talk about the compiler preprocessing? Why does the compiler do that?
3. What is the preprocessor?
4. Does the preprocessor create an actual file?
5. Why do some headers have quotes and others have angle brackets?
6. What directories will the compiler search when it is looking for header files?
7. So that's how it works for standard headers like `stdio.h`?
8. Can I create my own libraries?

a. Strictly speaking, the compiler just does the compilation step: it converts the C source code into assembly code. But in a looser sense, all of the stages that convert the C source code into the final executable are normally called compilation, and the gcc tool allows you to control those stages. The gcc tool does preprocessing and compilation.

b. Not necessarily, unless you are sharing code. You'll see more about this soon.

c. Yes.

d. Preprocessing is the first stage in converting the raw C source code into a working executable. Preprocessing creates a modified version of the source just before the proper compilation begins. In your code, the preprocessing step read the contents of the header file into the main file.

e. Yes. You can read through the `stdio.h` file on a Unix-style machine in `/usr/include/stdio.h`. It is probably in another directory on another machine style.

f. No, compilers normally just use pipes for sending the stuff through the phases of the compiler to make things more efficient. (4)

g. Strictly speaking, it depends on the way your compiler works. Usually quotes mean to simply look for a file using a relative path. So if you just include the name of a file, without including a directory name, the compiler will look in the current directory. If angle brackets are used, it will search for the file along a path of directories.

h. The gcc compiler knows where the standard headers are stored. On a Unix- style operating system, the header files are normally in places like /usr/local/include, /usr/include, and a few others.

Exercice XII - Questions/réponses

1. Considering Makefile, this seems like a lot of work just to compile source code. Is it really that useful?
2. If I write a makefile for a Windows machine, will it work on a Mac? Or a Linux machine?
3. Can I use `make` for things other than compiling code?
4. Can I create new datastreams? How many ?
5. Why is `FILE` in uppercase?

a. It's historic. `FILE` used to be defined using a macro. Macros are usually given uppercase names.

b. Yes, `make` is amazingly useful. For small projects, make might not appear to save you that much time, but once you have more than a handful of files, compiling and linking code together can become very painful.

c. Yes, additionnally to the 3 standard streams (`stdin, stdout, stderr`), you can open new datastreams by representing them by a pointer to a file. It depends on the operating system, but usually a process can have up to 256. The key thing is there's a limited number of them, so make sure you close them when you're done using them.

d. Because makefiles calls commands in the underlying operating system, sometimes makefiles don't work on different operating systems.

e. Yes. `make` is most commonly used to compile code. But it can also be used as a command-line installer, or a source control tool. In fact, you can use make for almost any task that you can perform on the command line.

Exercice XIII - Ecrire une simulation de la fonction `itoa`. On se contentera d'afficher le résultat plutôt que de retourner une chaîne de caractères, on pourra se limiter aux entiers positifs. On pourra se baser sur les fonction à nombre quelconque d'arguments (Que faudrait-il pouvoir assurer si on voulait renvoyer une chaîne de caractères ?)