Exercice 1

In this exercise, we rely on Unix system calls for process management and IO management.
You must use the following system calls: fork(), execvp(), pipe(), dup2(), close()

We assume that 2 binaries are accessible in the PATH:

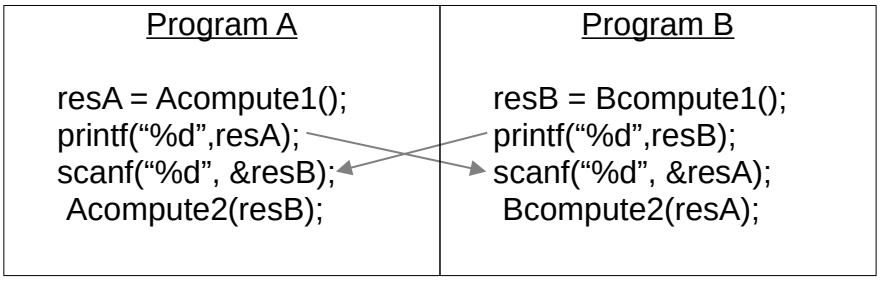
- **gendoc** allows to generate a document. The document is written on the standard output of the process.
- **print** allows to print document on the printer. The document is read from the standard input of the process.

You must write a program which:

- creates one process which executes the *print* binary
- the program then executes the *gendoc* binary
- the output from *gendoc* is transmitted to the input of *print*

Exercice 2

We want to implement the following communication pattern between two programs A and B.



Program A writes its result from Acompute1() to STDOUT, then it reads the result from B from STDIN in order to perform the second computation Acompute2(). Symmetrically, Program B writes its result from Bcompute1() to STDOUT, then it reads the result from A from STDIN in order to perform the second computation Bcompute2().

You must write a program (main.c) that will execute programs A and B, the STDOUT of program A being redirected to the STDIN of program B and the STDOUT of program B being redirected to the STDIN of program A.

Exercice 3

We want to implement a C program that implements the same behavior as the following shell command:

ps -ef | grep firefox | wc -l

Your program has to create 3 processes that respectively execute the ps, grep and wc binaries that are accessible from your path variable. The standard input and output of these processes are interconnected with pipes to implement the requested communication flow.
Here is a list of the Unix functions that you should use in your program:

```
int fork();
int execlp(char *file, char *arg, ...);
void perror(const char *s);
int pipe(int pipefd[2]);
int dup2(int oldfd, int newfd);
int close(int fd);
```