Name and surname: FIFO/pipe

Stage	1	2	3	4	Total
Points	5	7	6	7	25
Score					

L5: Roncevaux

Charlemagne, encouraged by the promise of peace from Marsile, the King of Spain, returns to France. At the rear guard, his procession is protected by his nephew, Count Roland. However, all of this is a plot by the treacherous Ganelon. Marsile's army engages Roland's forces in the Roncevaux Pass.

The battle is beautiful, fierce, and fierce,
The Franks strike eagerly and madly,
They cut off right hands, throats gurgle,
They slash garments to the living flesh,
Blood flows in streams across the grass

In this task, we will simulate the battle in the Roncevaux pass and do so in a way that generates new lines of 'The Song of Roland' (they will be in a similar style, though not very poetic). We will use processes and unnamed pipes for this purpose. Remember to close unused pipe ends and release resources, because:

This is what every worthy knight should do,
Who, in armor, on a noble steed,
Valiantly and bravely rides into battle.
Otherwise, he is not worth four denarii,
Better for him to go to a monastery,
And pray for our sins until death!

Stages:

1. $\boxed{\text{5 p.}}$ Upon starting, the program attempts to open two text files: `franci.txt` and `saraceni.txt`. If unsuccessful, it prints an appropriate message in the form `Franks/Saracens have not arrived on the battlefield`, depending on which file could not be opened, and then terminates. If successful, it reads and parses the contents of the files. The first line of each file contains an integer n, representing the number of knights. Each of the following n lines specifies the characteristics of a knight in the form of three space-separated fields: their name, HP, and attack. For each knight, the program prints the message `I am Frankish/Spanish knight <name>. I will serve my king with my <HP> HP and <attack> attack`. *Hint:* To simplify this, the starter code defines a constant `MAX_KNIGHT_NAME_LENGTH`.

2. $\boxed{\text{7 p.}}$ For each knight, the main thread creates an unnamed pipe and a child process using `fork`. Each knight should have an open read end of their own pipe and write ends to all knights of the opposing side—these will be used for simulating attacks. Close all unused pipe ends in all processes. Each child process prints the message from the first stage (the parent no longer does this) and then terminates. The parent waits for all child processes to finish. *Hint:* To ensure that all unused pipes are closed, you can use the helper function `count_descriptors`, which counts open file descriptors. Call it in each child process and print the result.

3. $\boxed{\text{6 p.}}$ Each knight's process executes the following operations in a loop:

    1. Reads all available bytes from its pipe (non-blocking read). Each byte represents a hit received, and its value is subtracted from the knight's remaining HP.
    2. Chooses a random enemy knight and attacks them. It randomly generates the strength of the attack—an integer S in the range $[0, attack]$—and writes it as a single byte to the target knight's pipe.
    3. Depending on the attack strength S, prints one of the following messages:
        - $S = 0$: `<name> attacks his enemy, however he deflected`

- $1 \leq S \leq 5$: `<name> goes to strike, he hit right and well`
- $6 \leq S$: `<name> strikes powerful blow, the shield he breaks and inflicts a big wound`

4. Randomly selects a number t in the range $[1, 10]$ and sleeps for t milliseconds.

Hint: To achieve random effects in child processes, use `srand(getpid())`. To improve the poetic nature of the output, you may add more phrases for attacking—but perhaps complete the last stage first.

4. $\boxed{\text{7 p.}}$ If a knight's HP falls below 0, they die. They print the message `<name> dies`, close all pipe ends, release all other resources, and terminate. A knight's process should detect when attacking an already dead enemy (error `EPIPE` or signal `SIGPIPE`), mark them as dead to avoid attacking them again, and retry against another enemy.

   Suggested implementation: Maintain a list of enemy knights along with a variable p representing the last known living enemy. If an attacked enemy i is found to be dead, swap them with the p-th enemy in the list and decrement p by 1. Other implementations (e.g., a linked list) will be accepted as long as they do not worsen complexity (e.g., randomly choosing an enemy repeatedly until finding a living one is inefficient and will not be accepted).

   If a knight determines that all their enemies are dead, they also terminate.