Programming 1 - Laboratories: Task 6

Josephus Flavius was a famous Jewish historian of the first century at the time of the Second Temple destruction. During the Jewish-Roman war he got trapped in a cave with a group of 40 soldiers surrounded by Romans. The legend has it that preferring suicide to capture, the Jews decided to form a circle and, proceeding around it, to kill every third remaining person until no one was left. Josephus, not keen to die, quickly found the safe spot in the circle and thus stayed alive.

Josephus Problem talks about a problem where there are N people standing in a circle waiting to be executed. The counting out begins from the first person in the circle and proceeds around the circle in a fixed direction. Every K^{th} person is eliminated from the circle until only one person remains. The original version of the problem corresponds to K = 3. For example, with 6 people in the circle (labelled 1 through 6), the order of elimination would be: 3, 6, 4, 2, 5 and person 1 would be the last person standing.

In this task you have to solve Josephus Problem (using circular singly-linked list of integers) for a given configuration (N and K) read from file. The entire elimination order should be written to text file. The proper solution can be checked at: http://webspace.ship.edu/deensley/mathdl/Joseph.html.

Before you start, copy the contents of `main.c` file into the source file where you will be solving this task. It contains the main function which will test your solution. You can uncomment parts of `main` function and add implementation of your own functions, but the existing code should remain unchanged unless the task description directs you to do so. Each part of the task involves implementing a specific function, but you are free to define additional helper function if you wish.

You should also download config.dat file containing sets of all possible configurations and place it in your project folder.

Part 1 (2 points)

Structure `config` (defined in `main.c`) has two integer members representing number of soldiers (`int number_of_soldiers`) and order of soldier elimination during counting out (`int kill_every_k`). An array of `config` variables containing 31 elements of possible configurations was stored (as a binary array) to the `config.dat` file.

Define a following function:

```
config read_config(const char *file_name_read, int idx);
```

which reads and returns the value of `config` variable at the specific index in the array form `config.dat` file. Function should read at most `sizeof(config)` bytes from the file (*hint*: use file positioning functions). If any errors occur the function should return `config` with number of soldiers less than 1.

In `main.c` create a preprocessor symbolic constant `IDX` representing your day of birth. That number will be used to select the configuration.

Part 2 (2 point)

Structure `soldiers_list` (defined in `main.c`) represents circular singly linked list, while structure `soldier` represents an element of `soldiers_list`.

Define a following function:

```
soldiers_list create_soldiers_list(int n);
```

which creates `n` list elements, labelled from `1` to `n` (use member `id` in `soldier` structure) and stores them in a circular linked list which should then be returned from the function. In case of allocation errors empty list should be returned instead.

Part 3 (2 point)

Write a function:

```
int eliminate_soldiers(const char* filename, soldiers_list* sl, int k);
```

which realize Josephus Problem counting out. In each step one element (every kth) from the list is removed until only one list element remains. The entire elimination order should be written to text file `filename`. Each line in the file should have information from single step, e.g.:

```
"In 1 step soldier 8 was killed.").
```

Function returns the initial position of the survivor (value of `id` member from the last remaining `sl` element). In case of any errors a value less than `1` should be returned instead.

Part 4 (2 point)

Ensure that your function free all allocated memory even if errors occur during list creation or elimination. Modify `main` so that file names (`config.dat` and `elimination_order.txt`) and the configuration index are given as command line arguments. Inform the user about the error if the number of arguments is incorrect or if the index is out of bounds.

Example

Example output of the program with IDX equal to 8:

```
Configuration for index 8
Number of soldiers: 41, kill every 8
The survivor is soldier number 14
```

Contents of elimination_order.txt:

```
In  1 step soldier  8 was killed.        In 21 step soldier 12 was killed.
In  2 step soldier 16 was killed.        In 22 step soldier 29 was killed.
In  3 step soldier 24 was killed.        In 23 step soldier  2 was killed.
In  4 step soldier 32 was killed.        In 24 step soldier 20 was killed.
In  5 step soldier 40 was killed.        In 25 step soldier 37 was killed.
In  6 step soldier  7 was killed.        In 26 step soldier 18 was killed.
In  7 step soldier 17 was killed.        In 27 step soldier 36 was killed.
In  8 step soldier 26 was killed.        In 28 step soldier 19 was killed.
In  9 step soldier 35 was killed.        In 29 step soldier  1 was killed.
In 10 step soldier  3 was killed.        In 30 step soldier 27 was killed.
In 11 step soldier 13 was killed.        In 31 step soldier 10 was killed.
In 12 step soldier 23 was killed.        In 32 step soldier  5 was killed.
In 13 step soldier 34 was killed.        In 33 step soldier 33 was killed.
In 14 step soldier  4 was killed.        In 34 step soldier 31 was killed.
In 15 step soldier 15 was killed.        In 35 step soldier 41 was killed.
In 16 step soldier 28 was killed.        In 36 step soldier  9 was killed.
In 17 step soldier 39 was killed.        In 37 step soldier 22 was killed.
In 18 step soldier 11 was killed.        In 38 step soldier 21 was killed.
In 19 step soldier 25 was killed.        In 39 step soldier  6 was killed.
In 20 step soldier 38 was killed.        In 40 step soldier 30 was killed.
```