HOMEWORK #2

SUBMISSION DUE DATE: 17/12/2019 23:00

INTRODUCTION

In this assignment, you are asked to implement a C program simulating the Nim game. The program simulates a game between a player and a computer, where the computer uses its own strategy, as explained below.

An executable implementing this assignment is provided to you, and your submission will be graded accordingly by matching your code's outputs on various inputs against the executable's outputs on the same inputs.

GAME PLAY AND ILLUSTRATION

The ordinary version of the game is between two players and played with N heaps of $h_1, h_2, \dots, h_N$ objects. The two players alternately take any number of objects from the top of one of the heaps. The winner is the one who empties all the heaps.

The following is an example of such a match. It has three heaps which initially contain three, four, and five tokens. Using its strategy, the computer wins.

EXAMPLE MATCH

Turn	Sizes of heaps h_1 h_2 h_3	Moves
1	3 4 5	
		Computer takes 2 objects from heap 1
2	1 4 5	
		User takes 3 objects from heap 3
3	1 4 2	
		Computer takes 1 objects from heap 2
4	1 3 2	
		User takes 1 objects from heap 2
5	1 2 2	
		Computer takes 1 objects from heap 1
6	0 2 2	
		User takes 1 objects from heap 2
7	0 1 2	
		Computer takes 1 objects from heap 3
8	0 1 1	
		User takes 1 objects from heap 2
9	0 0 1	
		Computer takes 1 objects from heap 3
10	0 0 0	Computer wins

PROGRAM IMPLEMENTATION

The program asks for the number of heaps N by the printing:

`"Enter the number of heaps:\n"`.

If the number of heaps is not a number between 1 and 32 the program prints `"Error: the number of heaps must be between 1 and 32.\n"` and terminates.

The program continues and asks for heaps sizes by printing
`"Enter the heap sizes:\n"`
If the i^{th} heap is not a positive number ($h_i > 0$) the program prints `"Error: the size of heap X should be positive.\n"` and terminates, where `X` is the corresponding heap index.

Before a turn is being played, the program checks if the heaps are empty in which case it announces the winner with `"You win!\n"` or `"Computer wins!\n"` and terminates.

If the heaps are still not empty, the program prints the status of the heaps with the text: `"In turn X heap sizes are: h1=Y h2=Z … hn=W.\n"`, where `X, Y, Z` and `W` are the actual corresponding values. Then either the user or the computer plays.

In a user turn, the program first prints the text `"Your turn: please enter the heap index and the number of removed objects:\n"`. The user then enters the heap index and the number of tokens to remove from it. You may assume the user only inputs integers. If the Input is valid, the program prints `"You take X objects from heap Y.\n"`. If the user input is valid (two integers) but illegal, the program prints `"ERROR: Invalid input.\nPlease enter again the heap index and the number of removed objects:\n"`. The program doesn't continue to the next turn until a valid input is provided. The number of tokens must be greater than 0.
You should read user input **in pairs**, i.e., read both values using *scanf("%d %d", …)* and only then check the return value, EOF, numbers read from the input, etc.

In a computer turn, the program computes the heap index and the number of objects to remove according the computer's strategy (see below), and prints `"Computer takes X objects from heap Y.\n"`.

ASSUMPTIONS

You may not assume anything regarding user input except what is explicitly written above. As in HW1, strange behavior may be observed for some inputs (7.3, 2x, etc.) due to using *scanf*. Match the provided executable.

At any point, if there is no more input (i.e., EOF) – the program should terminate without any additional output. If only a single input was provided in a user turn, ignore it, otherwise handle the input *and then* terminate. Use *`feof(stdin)`* to check for EOF. Match the provided executable when uncertain.

It is assumed that the user begins the game (unlike the example, above).

THE COMPUTER'S STRATEGY

DEFINITIONS

We denote the bitwise exclusive or (XOR) between two integers x, y as $x \oplus y$ (see below a detailed definition of XOR). Let h_k^t be the size of heap k in turn t. The **Nim-sum** is defined to be the XOR of all heaps, that is:

$$s_{nim}^t = h_1^t \oplus h_2^t \oplus \ldots \oplus h_n^t$$

The **Nim-sum of heap** i is the XOR of heap i with the Nim-sum:

$$h_i^t \oplus s_{nim}^t$$

Heap i is called a **winning heap** if its Nim-sum is less than its size, that is:

$$h_i^t \oplus s_{nim}^t < h_i^t$$

Otherwise the heap is called a **losing heap.**

COMPUTER STRATEGY

The computer strategy at turn t is:

1. If $s_{nim}^t \neq 0$ then find a winner heap $i' = \min\{i | h_i^t \oplus s_{nim}^t < h_i^t\}$. You may assume such a winner heap always exists. If several exists, choose the the lowest index.
    a. Remove $h_{i'}^t - h_{i'}^t \oplus s_{nim}^t$ objects from heap i'. The heap size will be equal to its Nim-sum: $h_{i'}^t \oplus s_{nim}^t$.
2. Otherwise (i.e., $s_{nim}^t = 0$) then find $i' = \min\{i \mid h_i^t > 0\}$ and take one object form heap i'.

An explanation of the algorithm's correctness can be found at the appendix at the end of the assignment.

IMPLEMENTATION HINTS AND GUIDELINES

XOR OPERATION

In the C programming language, operations can be performed on a bit-level using bitwise manipulation. An example of using the bitwise XOR operation is given below:

```
int x=10, y=12,w=100, z=0;
z= x^y;
z^=w;
```

In the above example, four integer variables were defined, x,y,w and z. In line 2 the value of z is set to $z = x \oplus y$ (now z=6) and in line-3 the value of z is set to $z = x \oplus y \oplus w$,i.e 98.

USING ARRAYS IN YOUR CODE

Use a constant-size array to represent the heaps (whose size is defined using #define). No need to dynamically allocate memory.

FILES PARTITIONING

Your code should be **clear and well documented** (i.e. contains comments and has reasonable variable names and function names). In this assignment you need to implement three source files:

1- main.c - This file must only contain the main function. Your main function should be as short as possible - no more than **60** lines of code.
2- sp_nim.c - This file contains functions that are directly related to the Nim game. For instance, you may want to implement the following functions in sp_nim:
   - A function that checks if there's a winner.
   - A function that helps you determines the computer's next move
3- main_aux.c - This file contains all auxiliary functions needed by the main function, such as:
   - A function that prints the heaps
   - A function that checks the validity of an input

Notice that all functions should be as short as possible. A function that has more than 60 lines should be shrunk by cascading the work wisely to other functions. Furthermore, you should define constants using macros (`#define`) and add comments to make your code more readable.

SUBMISSION GUIDELINES

Please submit a zip file named **id_assignment2.zip**. The zipped file contains the following:

- Header files - main_aux.h and sp_nim.h
- Source files - main.c main_aux.c and sp_nim.c

COMPILATION

To compile your code on NOVA you need to use the makefile provided in the assignment. Please notice that after invoking the 'make' command the resulting executable name is nim. Refer to homework #1 / tutorial 1 for more information.

CHECK YOUR CODE

An example executable is provided with the assignment. You should check your code by using the "diff" function as was shown in homework #1 and tutorial 1.

APPENDIX

The logic behind the definition of winner heaps is that reducing a winner heap from h_i^t to $s_{nim}^t \oplus h_i^t$ objects reduces the Nim-sum to zero in i.e. $s_{nim}^{t+1} = 0$. If on the other hand a player chooses a losing heap, the Nim-sum in the next turn will be different than zero (i.e. $s_{nim}^{t+1} \neq 0$).

There might be cases where a player might be forced to choose a losing heap: if the Nim-sum equals $s_{nim}^t = 0$ then $\left(s_{nim}^t \oplus h_i^t\right) = h_i^t$ and therefore every heap is a losing heap. The following theorem proves that this is the only scenario.

Theorem: There exists a winner heap in turn t iff the Nim-sum is different from zero, i.e. $s_{nim}^t \neq 0$.

Assume that turn t is a computer turn with a Nim-sum of $s_{nim}^t \neq 0$. The computer strategy is to find a winning heap, and to reduce its size from h_i^t to $s_{nim}^t \oplus h_i^t$. By doing so, the computer ensures that the user chooses a losing heap and the Nim-sum in the next computer turn is different from zero $s_{nim}^{t+2} \neq 0$.
The winner chooses the last set of tokens and must have a nim sum different than zero. Thus, if there exists a computer turn where the Nim-sum is different than zero, the computer wins.

For the automatic testing, you can assume the following assumptions:

1. If $s_{nim}^t \neq 0$, the computer must remove objects from the winner heap of minimal index i.e. $min\{i | h_i^t \oplus s_{num}^t < h_i^t\}$
2. If $s_{nim}^t = 0$ then the computer finds the non-zero heap of minimal index, $min\{i | h_i^t \neq 0\}$, and removes from it a single object.

GOOD LUCK