Assignment 7

Version 1.00 (last update: Nov. 16, 9:40)

Changes highlighted in yellow

Summary and Purpose

For this assignment, you will be working with both a hash-table and a graph data structure. You will be using these to create a misère nim player that never makes a mistake. You can play the game here:

https://www.archimedes-lab.org/game_nim/play_nim_game.html

Provided Files

You can find the following files on CourseLink:

1) A file called **`nim.h`** that contains **`struct`** definitions, and some function prototypes.
2) A file called **`nimhelp.c`** that contains two helper functions.
3) A file called **`alphanim.c`** that contains the code to play games of nim.

Do not include these files in your repository. Do not modify these files (since you are not including them in the repository and only the original files will be used during testing).

Deliverables

You will be submitting:

1) A file called **`nim.c`** that will be compiled to create an object file called **`nim.o`**.
2) A **`makefile`** that contains the instructions to compile your code.

The game of nim

Nim is a two-player strategy game played with row of matches (or stacks of coins, or piles of stones). Players take turns. During a player's turn they can remove one of more matches from a row. They cannot remove matches from more than one row; and they must pick up at least one match. The game ends when one player picks up the last match and, thereby, loses the game.

Data structures

Boards in the game will be represented by two variables. An integer, **`board_size`**, and an array of integers. **`board_size`** is used to indicate the number of indices in the array.

```
struct move {
  int row;
  int matches;
  int hash;
};
```

This structure represents a move in the game of nim. It indicates the row from which matches are taken, and the number of matches taken and a hash value that tells you the index of an array where the resulting board is stored.

```
struct node {
  int nimsum;
  int *board;
  int moves;
  struct move *move;
};
```

This structure represents a node in a graph describing the complete game of nim. **`board`** is an integer array recording the number of matches in each row (the number of rows is stored in a separate variable, **`board_size`**). **`moves`** is an integer that indicates the number of moves that are possible from the given board. **`move`** is an array of **`move structs`** of size equal to **`moves`**. **`nimsum`** is a variable describing the quality of the board for the current player (`0` is bad, `1` is good).

The functions and programs

```
int *new_board( int board_size );
```

This function should **malloc** an array of **board_size** integers and return a pointer to the array. If the **malloc** command fails, it should print a message to the standard error stream and **exit**.

```
struct node *mk_nim_hash( int max_hash, int board_size, int *start_board );
```

This function should malloc an array of size **max_hash struct node**s. It should initialize the nodes by setting the values of each **moves** value to **-1**, **move** value to **NULL**, **nimsum** value to **-1**. It should call the **hash2board** function (found in **nimhelp.c**) with the **board_size**, **start_board** and array index and store that variable in the **board** value at the same index in the array. If the **malloc** command fails, it should print a message to the standard error stream and **exit**.

```
void free_board( int *board );
```

This function should free a board array.

```
void free_nim_hash( int max_hash, struct node *nim_hash );
```

This function should **free** each **move** array and each **board** array in each element in the **nim_hash** and finally **free** the **nim_hash** array itself.

```
int *board_from_argv( int board_size, char **argv );
```

This function should create (**new_board**) a board array and initialize it with the integer equivalents (**atoi**) of the string array **argv**. The number of values to be converted in **argv** is equal to **board_size**.

```
int *copy_board( int board_size, int *board );
```

This function should return a pointer to a **new_board** whose values are initialized with exactly the same values as **board**.

```
int game_over( int board_size, int *board );
```

This function will return a value of 1 if the **board** contains exactly one match and 0 otherwise.

```
void join_graph( struct node *nim_hash, int hash, int board_size,
    int *start_board );
```

This function will recursively join the nodes of the graph beginning with the node at index hash in the nim_hash array. It should check whether the **moves** variable is -1; if not, it can exit (the moves for the node have already been computer). Otherwise, it should compute the

total number of possible moves that can be made for the given board, and allocate a **`move`** array within the node to store those moves. It should initialize each possible move with the appropriate row and matches values. Then it should compute the outcome of applying that move and store the **`hash`** value of the resulting move in the moves hash variable. Finally, it should call itself recursively on each destination node.

The Last 20%

```
int compute_nimsum( int board_size, int *board );
```

This function should compute the nimsum for a given **`board`**. The nimsum is defined as the **`bitwise exclusive OR`** of all the integer values in **`board`**, unless there are no values in **`board`** that are greater than **`1`**. In the latter case, the nimsum is the negation of the regular nimsum.

Add a call to the function **`compute_nimsum`** to your **`join_graph`** function to assign the **`nimsum`** value of each node visted.

You can write additional helper functions as necessary to make sure your code is modular, readable, and easy to modify. Place the functions only in nim.c (not nimhelp.c).

Testing

You are responsible for testing your code to make sure that it works as required. The CourseLink web-site will contain some test programs to get you started. However, we will use a different set of test programs to grade your code, so you need to make sure that your code performs according to the instructions above by writing more test code.

Your assignment will be tested on the standard SoCS Virtualbox VM (http://socs.uoguelph.ca/SoCSVM.zip) which will be run using the Oracle Virtualbox software (https://www.virtualbox.org/wiki/Downloads). If you are developing in a different environment, you will need to allow yourself enough time to test and debug your code on the target machine. We will NOT test your code on YOUR machine/environment.

Full instructions for using the SoCS Virtualbox VM can be found at: https://wiki.socs.uoguelph.ca/students/socsvm.

Makefile

You will create a makefile that supports the following targets:

`all:`

this target should generate the file **`nim.o`**.

`clean:`

this target should delete all **`*.o`** and executable files.

Each, **`*.c`** file should correspond to a rule in the **`makefile`** that generates a corresponding **`*.o`** file.

Additionally, there should be an individual rule for each executable file.

All compilations and linking must be done with the **`-Wall -pedantic -std=c99`** flags and compile and link **without any warnings or errors**.