CS 210 - Fundamentals of Programming I
Spring 2019 - Programming Project 3
20 points

Out: February 7, 2019
Due: February 14, 2019

Reminder: This is a programming assignment, and work on this assignment should be done individually. Assistance from other students is limited to questions about specific issues as noted in the syllabus.

Problem Statement
The game of Nim has existed in various forms since at least the 16th century (according to Wikipedia, so it must be true). It is fairly simple, yet still provides a challenge. Here are the rules for our version:

1. The game "board" consists of three heaps of stones initially containing 3, 4, and 5 stones each.
2. Players alternate turns removing stones from a single heap each turn.
3. A player may remove as many stones as they wish in a single move, but they must all come from the same heap.
4. The player that removes the last stone wins.

Our program will label the heaps as 'A', 'B', and 'C'. A move will be entered as the heap letter followed by the number of stones to be removed. For a move to be legal, the letter must be 'A', 'B', or 'C', and the number that follows must be greater than 0 and less than or equal to the number of stones remaining in that heap. See the sample run for some illustrated games.

As with Project 2, we want to create a program to allow a human to play this game against a computer opponent. This computer opponent will use a perfect strategy. (If it goes first, it will always win. If it goes second, and the human makes a mistake, it will win.) If you want to know more, the Wikipedia article provides more information than you will ever need. The program should allow the player to play as many games as she likes. For each new game, the loser of the previous game goes first. Therefore, if both players play perfectly, the games will be split evenly.

Program Specifications
Like the last project, you are given the program specification in the form of analysis and design ideas for a series of functions that must be implemented. Each function accomplishes one task needed for this program. You should write a function and then test it with a `main` program that ensures that the function works by itself. Once the function is working, move on to the next function, which will require a different `main` function to test it. Once all of the individual functions work, the final `main` function that actually will play the game is written.

Function: print_greeting
Analysis: No objects
Design: This function's task is simply to print out the greeting at the beginning of the program. See the sample run for the exact text format. It should only be run once, when the user first runs the program, not at the start of each game.

Function: user_wants_to_play_again
Analysis, Design, and Implementation covered previously.

Function: print_scoreboard

Analysis:

Objects	Type	Movement	Name
number of computer wins	int	received	comp_wins
number of user wins	int	received	user_wins

Design: This function's task is to print out the current number of wins by the computer and by the user. See the sample run for the exact text format.

Function: play_nim

Analysis:

Objects	Type	Movement	Name
Number of the player that starts	int	received	starting_player
Number of the player that won	int	returned	winner

Design: This function's task is to play one game of Nim with starting_player going first.. The design of much of this function is left up to you. However, here are some hints:

- You will need three local integer variables to hold the count for each of the three heaps throughout the game.
- You will need local variables to hold both the letter of the heap being chosen and the number of stones being removed.
- You will need a variable that tracks which player's turn it is.
- You will need to only allow legal moves by the user. A move is legal if:
  - The heap chosen is 'a', 'b', or 'c' (upper or lowercase)
  - The number of stones chosen to be remove is greater than zero and less than or equal to the number of stones in the chosen heap.

  Note that scanf can be used to read in more than one input. Just use a specifier for each input, and list the input variables in the same order after the specifier string. Also, remember that there should be a space before the %c specifier.
- If it is the computer's turn, you must call `get_computer_move` (see next function) to find the move that the computer has chosen. NOTE: While writing this function, you might bypass this step and simply allow the human to enter moves for both players in order to simplify debugging.
- The function loops until all of the heaps are empty. When this occurs, the player who just moved (i.e., the one who removed the last stone) is the winner **and the number of that player is returned to the calling function**.

Function: get_computer_move

Analysis:

Objects	Type	Movement	Name
Three heaps of stones	int	received	heap_a, heap_b, heap_c
Computer's chosen heap	char	passed back	chosen_heap
Computer's chosen number of stones	int	passed back	number_to_remove

Note: passed back parameters will be covered in class on Tuesday, February 12. They are in Chapter 6 of the textbook and are called "output parameters" there.

Design: The winning strategy for this game relies on counting and canceling out powers of two. Again, read the Wikipedia article if you are interested in the details. The short answer is that we can find what is called the Nim-number of the game by computing the bitwise-exclusive-or (XOR) of all the heaps. Fortunately, C has an operator that can do this (`^`) (not exponentiation). If we can make the Nim-number of the game be 0 after every move, we are guaranteed to win.

Here is the algorithm:

1. Calculate the Nim-number of the game by XOR-ing all the heaps together. (e.g. **`nim_number = heap_a ^ heap_b ^ heap_c;`**)
2. If that number is already 0, we do not have a good move, so simply remove 1 stone from the first heap that has one available. (We are going to try to stall for time).
3. Otherwise, we need to find a move that changes the Nim-number to 0. To do this we perform the following until we find a heap that works:
   (a) If **`(heap ^ nim_number) < heap`**, for any heap then this is the heap that we will remove stones from.
   (b) The number of stones to remove will be **`heap - (heap ^ nim_number)`**

Main Program

Analysis:

Objects	**Type**	**Name**
Number of wins by the computer	int	comp_wins
Number of wins by the user	int	user_wins
Starting player number (initially the user)	int	starting_player

Design: The main function for this program ends up being fairly simple and similar to the Guessing Game. Most of the complicated logic is in the other functions. Main needs to keep track of the number of wins for each player and which player is the starting player for the next game. It needs to repeatedly call the play_nim function, record the winner, print out the current scoreboard, and ask the user if they want to play again (by calling that function) until the user is done playing.

Assignment

Write a C program that implements the game of Nim. **Your program must follow the specifications given above to earn full credit.** That is, your program must define and use at least the specified functions. (The names of functions and variables do not have to be exactly the same.)

The output of the program must conform exactly to the following example run (now that we have loops, there is only one sample run; user input shown in **bold**). Note there is blank line before each player's move, before the scoreboard display, and before the "Do you want to play again..." prompt, and there is one space after the first sentence punctuation in a two sentence line of output. (E.g. in the error messages.) And as usual, there must be a newline after the last line of output.

```
Welcome to the Ancient Game of Nim
Player 1 is you (the human)
Player 2 is me (the computer)
Player 1 goes first this time!

Player 1
Heaps:
A: 5
B: 4
C: 3
Enter the letter of the heap and number of stones to remove: c2

Player 2
Heaps:
A: 5
B: 4
C: 1
COMPUTER moves a1

Player 1
Heaps:
A: 4
B: 4
C: 1
Enter the letter of the heap and number of stones to remove: B2

Player 2
Heaps:
A: 4
B: 2
C: 1
COMPUTER moves a1

Player 1
Heaps:
A: 3
B: 2
C: 1
Enter the letter of the heap and number of stones to remove: D4
Illegal move! Try again.

Player 1
Heaps:
A: 3
B: 2
C: 1
Enter the letter of the heap and number of stones to remove: c2
Illegal move! Try again.

Player 1
Heaps:
A: 3
B: 2
C: 1
Enter the letter of the heap and number of stones to remove: c1
```

```
Player 2
Heaps:
A: 3
B: 2
C: 0
COMPUTER moves a1

Player 1
Heaps:
A: 2
B: 2
C: 0
Enter the letter of the heap and number of stones to remove: b1

Player 2
Heaps:
A: 2
B: 1
C: 0
COMPUTER moves a1

Player 1
Heaps:
A: 1
B: 1
C: 0
Enter the letter of the heap and number of stones to remove: b1

Player 2
Heaps:
A: 1
B: 0
C: 0
COMPUTER moves a1
Player 2 wins!

* * * * * * * * * * *
Current Score:
Player 1 (Human):    0
Player 2 (Computer): 1

Do you want to play again? (y/n) Y
Player 1 goes first this time!

Player 1
Heaps:
A: 5
B: 4
C: 3
Enter the letter of the heap and number of stones to remove: a5
```

```
Player 2
Heaps:
A: 0
B: 4
C: 3
COMPUTER moves b1

Player 1
Heaps:
A: 0
B: 3
C: 3
Enter the letter of the heap and number of stones to remove: b3

Player 2
Heaps:
A: 0
B: 0
C: 3
COMPUTER moves c3
Player 2 wins!

* * * * * * * * * * *
Current Score:
Player 1 (Human):    0
Player 2 (Computer): 2

Do you want to play again? (y/n) p
Bad input! Try again.

Do you want to play again? (y/n) n
```

REMINDER: Your program must compile for it to be graded. Submissions that do not compile will be returned for resubmission and assessed a late penalty. Submissions that do not substantially work also will be returned for resubmission and assessed a late penalty.

Follow the program documentation guidelines in the *C Programming Style Guideline* handout. As stated in the syllabus, part of the grade on a programming assignment depends on how well you adhere to the guidelines. The grader will look at your code and grade it according to the guidelines. **Be sure to run the Source code formatter (Astyle) plugin before you submit your program.**

What to Submit

Electronically submit a zipfile containing **`main.c`** (only) as explained in class and in the handout *Submission Instructions for CS 210*. The submission system will start accepting assignments no earlier than the evening of Sunday, February10 . Reminders: you may submit as many times as needed, and only the last submission will be graded. All programming assignments are due by 11:59pm.