COMP 2401B -- Assignment #2

<u>Due:</u> Thursday, October 18, 2018 at 12:00 pm (noon)

Goal

Timmy Tortoise and Harold the Hare have been charged with rescuing a baby Dragon being kept captive in an underground bunker by a band of pirates. In order to compute their chances of a successful rescue, Timmy Tortoise wants to use a program to simulate the battles between our heroes and those pirates. Using the VM provided for the course, you will implement a C program that simulates Timmy and Harold fighting each pirate one at a time. The program will execute 100 runs of this simulation, it will determine the winner each time, and it will collect statistics on survival rates. The winner of each simulation will be the survivor(s), either one of the heroes, or both heroes, or the pirates if both heroes are dead.

Learning Outcomes

You will practice problem solving and designing modular functions to implement a solution. You will work with structures in C, as well as pointers and dynamically allocated memory.

Instructions

1. Data structures

Your program will define the following data types as structures:

- the `HeroType` structure will contain a hero's name (as a string), as well as indicators of his strength, his armour, and his health, all represented as integers
- the `PirateType` structure will contain a pirate's strength and health indicators
- the `ArrayType` structure will be a *collection structure* which contains a primitive (regular) array of `PirateType` pointers; the `ArrayType` will also need to store the current number of pirates in the array

Your program will initialize one `HeroType` structure for Timmy, and another one for Harold, as follows:

- Timmy's strength is set at 5, his armour at 5, and his health indicator begins at 30 points
- Harold's strength is set at 7, his armour at 3, and his health indicator begins at 30 points

For each run of the simulation, a different `ArrayType` structure will be initialized to hold the pirates:

- the array structure will be filled with 10 dynamically allocated `PirateType` structures
- each pirate will have a strength as a randomly generated number between 3 and 6 (inclusively)
- each pirate will have a health indicator that begins at 20 points
- each pirate will be pushed to the back (the end) of the `ArrayType` structure

2. Program behaviour

You will design modular functions to implement the following functionality. Your program will:

- initialize the counters that will track the number of times each hero wins (the number of runs where Timmy lives and Harold dies, and vice-versa), the number of times that both heroes win (the runs where they both live), and the number of times that the pirates win (the runs where both heroes die)
- loop for 100 runs of the simulated battle; in each iteration:
  - a new pirate array is initialized with fresh pirates, as described above
  - both heroes' health indicators are reset to 30 points
  - loop as long as at least one hero is still alive and the pirate array is not empty:
    - one hero (alternate between Timmy or Harold) fights the next pirate at the back of the pirate array, according to the fight rules described below, until one of them is dead
    - if the pirate dies, he is popped off the array, and the other hero fights the next pirate
    - if the hero dies, the other hero fights the same pirate
  - determine the winner of the battle: both heroes if both survive, Timmy if he survives and Harold dies (and vice-versa), or the pirates if both heroes die
  - update the winner counters according to the result of the battle
- after all 100 runs, print the statistics indicating the winning rates of each hero alone, of the two heroes together, and of the pirates

Note: You can use the random number generator function that was provided for Assignment #1

3. Fight rules

A fight between one hero and one pirate is simulated as follows:

- loop as long as one of the fighters is still alive; in each iteration:
  - the hero attacks the pirate:
    - the pirate's health is decreased by the hero's strength
    - if the pirate's health becomes zero or less, he dies
    - if the pirate dies, the hero celebrates, his health increases by 3 points, and the fight is over
    - if the pirate survives, he attacks the hero
  - the pirate attacks the hero:
    - the pirate's temporary strength at each turn is determined as a random number between the pirate's strength and double his strength (inclusive)
    - the hero's health is decreased by the pirate's temporary strength, minus the hero's armour
    - if the hero's health becomes zero or less, he dies, and the fight is over

Constraints

- your program must be correctly designed and separated into modular, reusable functions
- your program must reuse functions everywhere possible
- your program must perform all basic error checking
- your program must be thoroughly documented, including each function and parameter
- compound data types must always be passed by reference
- all dynamically allocated memory must be explicitly deallocated
- do not use any global variables

Submission

You will submit in *cuLearn*, before the due date and time, one `tar` or `zip` file that includes the following:

- all source code, including the code provided, if applicable
- a readme file that includes:
  - a preamble (program author, purpose, list of source/header/data files)
  - the exact compilation command
  - launching and operating instructions

Grading (out of 100)

Marking components:

- 50 marks: correct declaration and initialization of data structures
  - 5 marks: correct declaration and initialization of `HeroType`
  - 5 marks: correct declaration and initialization of `PirateType`
  - 20 marks: correct declaration and initialization of `ArrayType`
  - 10 marks: correct pushing onto `ArrayType`
  - 10 marks: correct popping of `ArrayType`
- 50 marks: correct program behaviour
  - 10 marks: correct implementation of heroes taking turns with each pirate
  - 20 marks: correct implementation of the fight between one hero and one pirate
  - 10 marks: correct computing and printing of statistics
  - 10 marks: correct cleanup of data

Deductions:

- Packaging errors:
  - 100 marks for an incorrect archive type that is not supported by the VM
  - 50 marks for an incorrect archive type that is supported by the VM
  - 10 marks for a missing readme
- Major programming and design errors:
  - 50% of a marking component that uses global variables
  - 50% of a marking component that is incorrectly designed
  - 50% of a marking component that doesn’t pass compound data types by reference
- Minor programming and design errors:
  - 10% for consistently missing comments or other bad style
  - 10% for consistently failing to perform basic error checking
  - 10% for memory leaks
- Execution errors:
  - 100% of a marking component that cannot be tested because it doesn’t compile or execute in VM
  - 100% of a marking component that cannot be tested because it’s not used in the code
  - 100% of a marking component that cannot be proven to run successfully due to missing output