AERO 552: Aerospace Information Systems
Homework 1

Out Monday, September 4th, 2023

Due on CANVAS Thursday, September 14th, 2023, **11:59pm**

Instructions

Your solution should be submitted as the following files:

- a file `hw1.pdf` containing your solution to the handwritten problems;
- file `navigation.cpp` containing your solution to Exercise 1;
- file `hanoi.cpp` containing your solution to Exercise 1;
- files `Tree.h`, `Tree.cpp` and `tree_main.cpp` containing your solution to Exercise 3;
- file `life.cpp` containing your solution to Exercise 4, as well as a few test case in the form of `.dat` files;

For the coding exercises, remember the following guidelines:

- **Do not use any C++ library unless explicitly authorized, except `iostream`, `fstream`, `cmath`, and `string`**
- High-level discussion of problems and directions with other students is fine, but exchanging solutions or code is not. If in doubt, please ask. **If you discussed with another student, please indicate so explicitly in your homework, with their name.**
- All code from this homework will be graded on CAEN Linux using the `g++` compiler – please test it on CAEN prior to submission.
- Include your code along with the examples you have tested your code on. Examples would typically be in a function `main`. Please **thoroughly test your code**: part of your grade will be based on how well you have tested your code, including corner cases. Moreover, many errors can be caught with good testing.
- Please submit code that compiles (on Linux) **without any warning or error**, and runs right away. We will take off points if your code does not run or if it produces warnings.
- Be careful with pointer and memory management, and do not forget to `delete` your memory, even in your test cases. Points will be taken off for memory leaks (forgetting to `delete`).
- Please do not reference or utilize any online code.

Getting to know you (3 points)

On the first page of your homework, please include a recent picture of yourself, along with why you are taking this class, why any information you'd like the instructor to know.

Exercise 1 (code): Aircraft Navigation (20 points)

In preparation for his historic flight across the Atlantic Ocean, Charles Lindbergh would like to navigate from Ann Arbor (point A, latitude `latA` and longitude `longA`) to Boston (point B, latitude `latB` and longitude `longB`). He has tasked you with computing his heading θ (in degrees) and flying distance d (in nautical miles (nmi)). Using the Haversine formula for the loxodrome, the (approximate) values of d and θ are given by the following formulas, using intermediate value a and the radius of the earth $R = 3{,}440\,\text{nmi}$:

$$
\begin{aligned}
a &= \sin^2\left(\frac{\texttt{lat2} - \texttt{lat1}}{2}\right) + \cos(\texttt{lat1})\cos(\texttt{lat2})\sin^2\left(\frac{\texttt{long2} - \texttt{long1}}{2}\right) \\
d &= 2R\arctan2\left(\sqrt{a}, \sqrt{1-a}\right) \\
\theta &= \arctan2\big(\sin(\texttt{long2} - \texttt{long1})\cos(\texttt{lat2}), \\
&\qquad\qquad \cos(\texttt{lat1})\sin(\texttt{lat2}) - \sin(\texttt{lat1})\cos(\texttt{lat2})\cos(\texttt{long1} - \texttt{long2})\big)
\end{aligned}
$$

Write a C++ function
`void fly(double latA, double longA, double latB, double longB)`
that computes the proper heading and distance between point A and point B, for any pair of points on the planet. Its execution should print a message looking like:
`Fly heading 290.15 degrees for 630.34 nautical miles.`
Note: Latitude and longitude are given in degrees, and your output should be in degrees too. Beware that all C++ mathematical functions work using radians.
Note: Your program should correctly reject incorrect latitude and longitude values. Additionally, it should never crash, and the heading it prints should be between 0 degrees (included) and 360 degrees (excluded).

Exercise 2 (code): The tower of Hanoi (20 points)

The tower of Hanoi (Fig. 1) is a game consisting of three rods A, B and C, and a number `n` of disks of different sizes, initially stacked on rod A in decreasing order of size (the largest at the bottom and the smallest at the top). The goal of the game is to move the stack of disks to rod C, respecting the following rules:

- only one disk can be moved at a time;
- a move consists of taking a disk on top of one rod and placing it on top of another rod;
- no disk may be placed on top of a smaller disk.

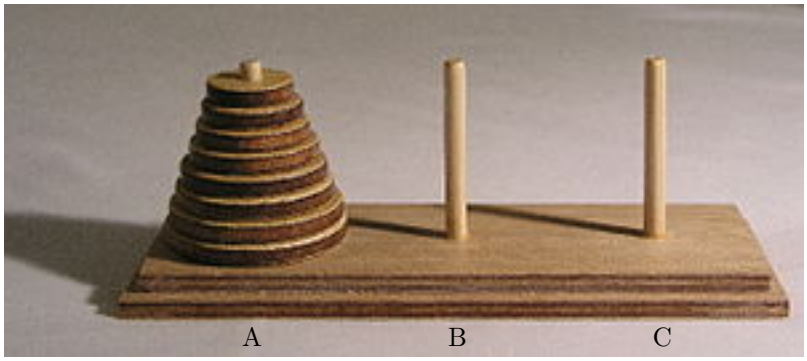


Figure 1: The tower of Hanoi. Source: Wikipedia

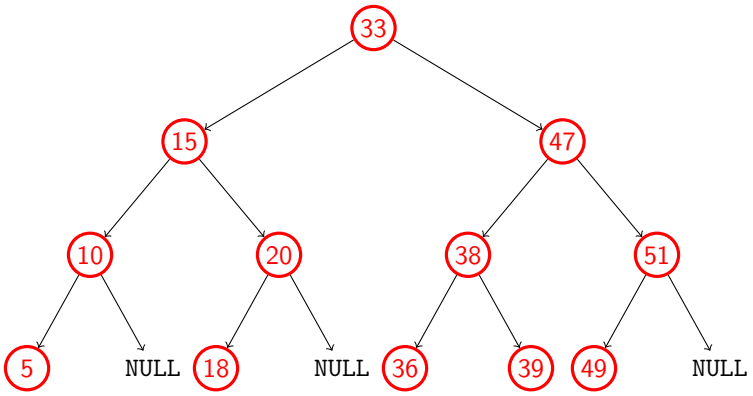
Write a function `void hanoi(int n)` that, given a number `n` of disks initially on rod A, outputs the list of steps necessary to move the stack to rod C. For example, for `n` = 3, the program should output the following:

```
Make a move from A to C
Make a move from A to B
Make a move from C to B
Make a move from A to C
Make a move from B to A
Make a move from B to C
Make a move from A to C
```

Exercise 3 (code): Binary trees (40 points)

For this exercise you will need to first read about object-oriented programming and classes in C++, at the end of chapter 1 of the lecture notes (many other resources are also available online).

A binary tree can be used to efficiently represent a set of integers. The idea is to store an integer r at the root of the tree, all the integers less than r in the left subtree, and all the integers greater than r in the right subtree. Visually, an example of such a tree looks like:



A possible datastructure to store such trees is:

```
struct treeNode {
  int root;
  struct treeNode* left;
  struct treeNode* right;
}
```

Implement a C++ class `Tree` holding such a set of integers. Properly hide the private data (the datastructure above). Do not worry about keeping the trees balanced. Provide the following member functions:

1. A default constructor making an empty set;
2. A copy constructor;
3. A method `void insert(int i)` inserting an element in a set;
4. A method `void erase(int i)` erasing an element from a set; does nothing if the set does not contain `i`;
5. A method `bool member(int i)` testing if `i` is a member of the set;
6. A method `int size()` returning the size of the set;
7. A method `int* toarray()` returning the set as an array, stored in increasing order;
8. A `<<` operator that prints all members of the set to the screen, in increasing order and separated by spaces (refer to the `Vector` example in the lecture notes for an example of how to do that). Do not call `toarray` to do this;
9. A method `Tree treeUnion(Tree a, Tree b)` computing the union of two sets;
10. A method `Tree intersection(Tree a, Tree b)` computing the intersection of two sets.

Submit your code in `Tree.h` and `Tree.cpp`, and your test examples in `tree_main.cpp`.

Exercise 4 (code): The game of life (25 points)

Developed by the mathematician John Horton Conway, the game of life consists of a rectangular $n \times m$ grid of cells, where each cell can be alive or dead. Every cell interacts with its eight neighbors (horizontally, vertically, diagonally adjacent), and at each step of time:

- a cell with fewer than two live neighbors dies (underpopulation);
- a cell with two or three live neighbors continues to live;
- a cell with four or more live neighbors dies (overpopulation);
- a dead cell with exactly three live neighbors becomes alive (reproduction).

We will represent a grid as a text file with n lines, each of length m. Each line consists of the letter O for an alive cell and the standard space character (written ␣ in this document) for a dead cell. Write a function `void life()` computing the evolution of the cells. Your function should expect an initial pattern in the file `life.dat`. An example `life.dat` is provided for a simple example. An invalid pattern (for example, the lines are not all the same length, or a line contains an illegal character) should be immediately rejected. After that, your program should compute the live cells at the next step, print the new pattern, then wait for the user to press Enter. After the user presses Enter, it computes yet another step, etc. until the user writes `quit`.

For example, if the file contains the pattern:

```
␣␣␣␣␣
␣␣O␣␣
␣␣O␣␣
␣␣O␣␣
␣␣␣␣␣
```

your program should initially output:

```
␣␣␣␣␣
␣␣␣␣␣
␣OOO␣
␣␣␣␣␣
␣␣␣␣␣
Press <Enter> to continue or type quit
```

then continue on after you press Enter.

Please test your code extensively. A number of interesting patterns and their evolution can be found on the Wikipedia page "Conway's Game of Life".

Bookkeeping (2 points)

In your `hw1.pdf`, indicate in a sentence or two how much time you spent on this homework, how difficult you found it subjectively, and what you found to be the hardest part. How deeply do you feel you understand the material it covers (0%–100%)? Any non-empty answer will receive full credit.

Indicate in a sentence or two:

1. how much time you spent on this homework;
2. how difficult you found it subjectively;

3. what you found to be the hardest part;
4. how deeply you feel you understand the material it covers (0%–100%).

Any non-empty answer to each of these questions will receive full credit.