C++ Exercise 7: Aliens

Deadline: Monday, November 9, 2020 by 23:59:59 EET Timezone.
How to submit: Check your code into your GitHub repository, the default `master` branch, tag it as `ex07submit` (all lowercase, no dashes).
Grading: This exercise is worth 100‰(or 10%) of the total grade.
Objectives: Using pointers, arrays, user-defined classes (but no external libraries like STL or Boost) complete the node traversal exercise described below: Read input from STDIN, write output to STDOUT and report error conditions, if input data has inconsistencies. **Performance targets:** RAM Memory: 4 MiB. Run-Time on a single input file: 0.1 seconds.
If your program is correct, but not very efficient, you can still get most of the credit (about 70% of the testcases will be small: no more than 100 nodes of the input data). But the performance targets should guide you against choosing some outrageously inefficient algorithms. **UML Class Design:** During the first week we suggest that you develop a class design (preferrably as a UML diagram or, maybe, a set of CPP header files) showing the function prototypes that you will use. And some testcases (Check2 or anything else) to test these functions. It should be submitted by **November 2** (a separate folder in ORTUS). During the second week you can write all the code to implement this class design (you can make changes, if you find out that you do not like your class design).

Description
In 2020 humans sent an autonomous robot *Perseverance* to Mars. (The goals were explained in their press-release as follows: ”The program's ongoing series of missions is helping us answer key questions about the potential for life on Mars. While previous missions have helped us look for signs of habitable conditions in ancient times, *Perseverance* will take it one step further by searching for signs of past microbial life itself.”) Soon *Perseverance* noticed various signs of life on Mars. They took images of green creatures who could change their location. After trying to hide for some time, the creatures (subsequently called *aliens*) started to communicate with our civilization.
To address culture differences and avoid misunderstandings, earthlings became interested in the everyday life of aliens. As they found out, aliens have asexual reproduction: every alien can have no more than two children. Every child is either left-handed or right-handed. Additionally, if an alien has two children, they are necessarily of different types: one of them is left-handed, and the other one is right-handed. Reproduction can happen soon after birth, many generations can live simultaneously. Since aliens have very large families, every alien maintains close relationship with its *favourite relatives* (every alien has up to 2 such relatives).
Earthlings wanted to find out, how to identify the two (or fewer) favorite relatives for the given alien. First, we pick some alien without living parent. Then draw the family tree (the tree of parent-child relationships with the given alien as the root). Every alien in this family tree has two favorite relatives immediately preceding him and immediately following him in the *inorder* DFS traversal of that tree. Left-handed aliens are shown to the left of their parent, the right-hinded ones - to the right.
Your task is to build an efficient program that receives the genealogy data for one or more family trees, and it finds the two favorite relative aliens for any given alien upon receiving a query.
It is known that no more than 10 000 aliens are currently alive. Every alien has a unique number [1..10 000].

Input
We add nodes one at a time. Commands can come at any order (but if we add a child, its parent must be added – otherwise it should be reported as an error). In-between the commands (that gradually build a "forest" of one or more trees) we can add query commands to find the *favorite relatives* for a given node in the tree that has been built so far.
The input file contains one of these five commands that build a collection of genealogy trees:

1. Specify a *Top living ancestor* of an alien family tree (it has no living parents).
2. Specify the *Left-handed child* for a parent.
3. Specify the *right-handed child* for a parent.
4. Query for the favorite relatives of a given alien.
5. Finish your work.

The syntax for all these commands looks like this:

```
T Ancestor
...
L Parent Child
...
R Parent Child
...
? Alien
...
F
```

- `Ancestor` is a number for an alien that is a *top living ancestor* of some genealogy tree. (`Ancestor` may stay the only node in his tree; then both his favorite relatives are nonexistent).
- `L` is the command to specify the left-handed `Child` for a `Parent`.
- `R` is the command to specify the right-handed child for a parent.
- In the query command `Alien` is the alien identificator in the request to find the favorite relatives.
- Any identificator for a an alien (`Ancestor`, `Parent`, `Child`, `Alien`) is an integer number from the interval [1; 10 000])

The input data is valid - regarding the format and the limitations defined above. Some tree editing commands may refer to nodes that cannot be inserted (such commands should be ignored and an error message printed). If a tree editing command succeeds, then a new node is added.

Output
Depending on the input file the output file contains output for every query command and also every command that ends in failure. (Successful tree editing commands are silently executed, they do not produce any output.)

```
PrevFav NextFav
...
error0
...
error1
...
error2
...
error3
...
error4
...
error5
```

Explanations for this syntax:

- `PrevFav` is the immediately preceding alien (in *inorder* DFS traversal order) for the given `Alien`, when we run the command `? Alien`. Therefore, it is one of the two favorite relatives of the `Alien`. If there is no previous node, output `0`.
- `NextFav` is the immediately following alien (in *inorder* DFS traversal order) for the given `Alien`, when we run the command `? Alien`. If there is no next alien, output `0`.
- `error0` – alien with identification number ‘Alien‘ does not exist in the family tree, when we run the command ‘? Alien‘.
- `error1` – `Parent` and `Child` are the same.
- `error2` – `Parent` does not exist in any family tree.
- `error3` – `Child` is already used in some family tree.
- `error4` – `Parent` already has a left-handed child in the family tree.
- `error5` – `Parent` already has a right-handed child in the family tree.
- `error6` – `Ancestor` cannot start a new tree, since is already used in some family tree.

If a command causes multiple errors at once, print the one with the smallest number.

Sample Input and Output

Sample input **test01in.txt**:

```
T 7
L 7 2
R 7 9
L 9 5
? 2
R 5 6
L 2 4
L 5 1
? 5
? 8
L 1 4
F
```

Expected output **test01expected.txt**:

```

0 7
1 6
error0
error3
```

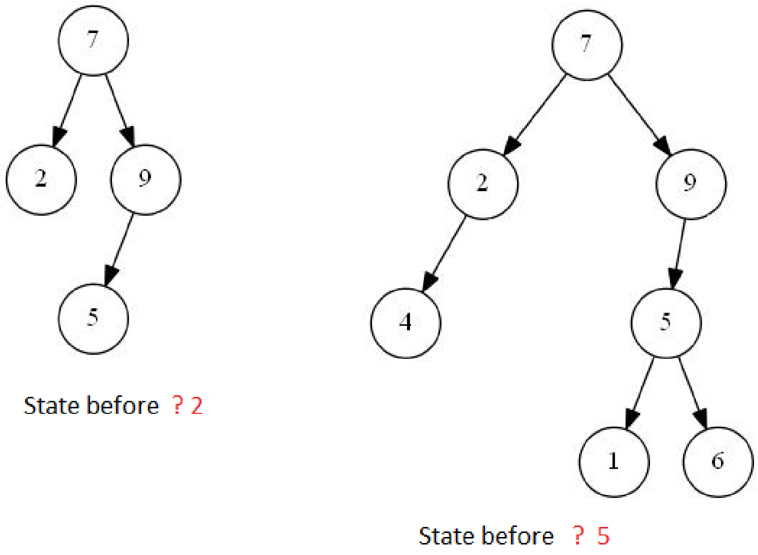

Figure 1: The data structure at two moments in time.