CS302

Assignment 9: Ancestors

Description

Family genealogy and ancestry is such a mystery. Even with the family tree it can be quite the job to map out everyone's relative. Luckily for this assignment we only need to map out all the ancestors of each relative, although mapping out cousins and second cousins would be fun but would be a monster of an assignment and I already fried everyone's brain with assignment 7, so we'll just output ancestors only in the family tree. For this assignment, you can use any STL class since we've spent all semester with pointers and custom containers. The input in this file will be plain text file with an edge detail per line, where each line contains

```
from -> to
```

this line denotes there is a directed edge from "from" to "to". You can use std::getline and std::stringstream to parse each line. You would need to create an out-neighbor adjacency list. I would recommend declaring the following structure

```
std::unordered_map< std::string, std::list<std::string> > adjList;
```

This maps a person's name to a linked list of children (or its direct descendants). So if you want to add descendants to "Bob" you can write the following code

```
adjList["Bob"].push_back("Janice");
adjList["Bob"].push_back("Marty");
adjList["Bob"].push_back("Patrick");

//If you want to traverse the descendants
//of "Bob", then you can have
for (auto descendant = adjList["Bob"].begin();
    descendant != adjList["Bob"].end(); descendant++)
{
  std::cout << *descendant << "\n";
}
```

In this scenario, `descendant` would be a linked list iterator, its type would be `std::list<std::string>::iterator`. Once you have read in the file and created the adjacency list, you will need to write an algorithm similar to DFS traversal, to be able to gather all the ancestors of each node in the graph (so you will need to run DFS multiple times). The start node(s) will be the node(s) that have no ancestors, and from those node(s) you traverse the graph until you find your target relative and incrementally build your list of ancestors. You might need a visited array but since the vertices are labelled by names, you would have

```
std::unordered_map<std::string, bool> visited;
```

You may also need to declare more objects as well. This program would output a sorted list of all ancestors for each person (if a person has no ancestors then output `"None"` without the quotes). Also you need to output each person sorted as well, otherwise code grade would have issues. You can use `std::sort` function from the `#include <algorithm>` library. So if you have a vector

```
std::vector<std::string> names;

//assuming you inserted several strings into names
//the code below will sort the list

std::sort(names.begin(), names.end());
```

This write up is relatively short since I'm explaining in more detail in the supplemental video.

Sample Run

```
$ ./a.out

Enter file: fam01.txt

Relative Name: five
List of ancestors
one
three
zero

Relative Name: four
List of ancestors
two
zero

Relative Name: one
List of ancestors
None

Relative Name: seven
List of ancestors
one
three
two
zero

Relative Name: six
List of ancestors
four
one
three
two
zero

Relative Name: three
List of ancestors
one
zero

Relative Name: two
List of ancestors
None

Relative Name: zero
List of ancestors
None
```

Specifications

- No specifications

Submission

Submit the source files to code grade by the deadline

References

- Supplemental Video https://youtu.be/i98s6pIEb70