Background

A young entrepreneur in your RHET 105 class has approached you with an idea that they claim is "the next big thing!" Since you are a rockstar CS 128 student, you've been tasked with building IlliniBook—a way to find and connect with fellow Illini based on shared interests.

The app should not only know who is connected to whom, but also how they are connected. When two people are directly connected, they have a label on their relationship, represented by a `std::string`. This relationship can be a club, activity, class, job, etc.

You don't want to deal with fancy things like "databases" and "the cloud" just yet, so all users have their information stored in two different .csv files.

After processing peoples' profiles, you'll need to be able to check if two people are related (directly and indirectly) and check if two people are related through a specific relationship type. You should also be able to find the number of groups in your social network!

The Illini Book Constructor

In the IlliniBook Constructor, you are supposed to read from two CSV files:

The first CSV file provides a list of UINs as `int`eger values:

```
UIN (int)
UIN_1
UIN_2
...
UIN_N
```

The second CSV file provides a list of relationships in a format that looks like this:

```
UIN_A,UIN_B,Relationship (int, int, std::string)
UIN_A_1,UIN_B_1,Work
UIN_A_2,UIN_B_2,Sports
...
UIN_A_N,UIN_B_N,relation_N
```

For emphasis, the UINs are *signed* `int`eger values (including `0`).

To help you parse the relationships file, I recommend performing a formatted read of types `int`, `char` (i.e., the ","), `int`, `char` (i.e., the ","), `std::string`. However, we do provide the function `utilities::Split` in `utilities.cc`, though you shouldn't have to use it.

Note: we will not include headers in the csv file. Please see the *example* section of this writeup for more example.

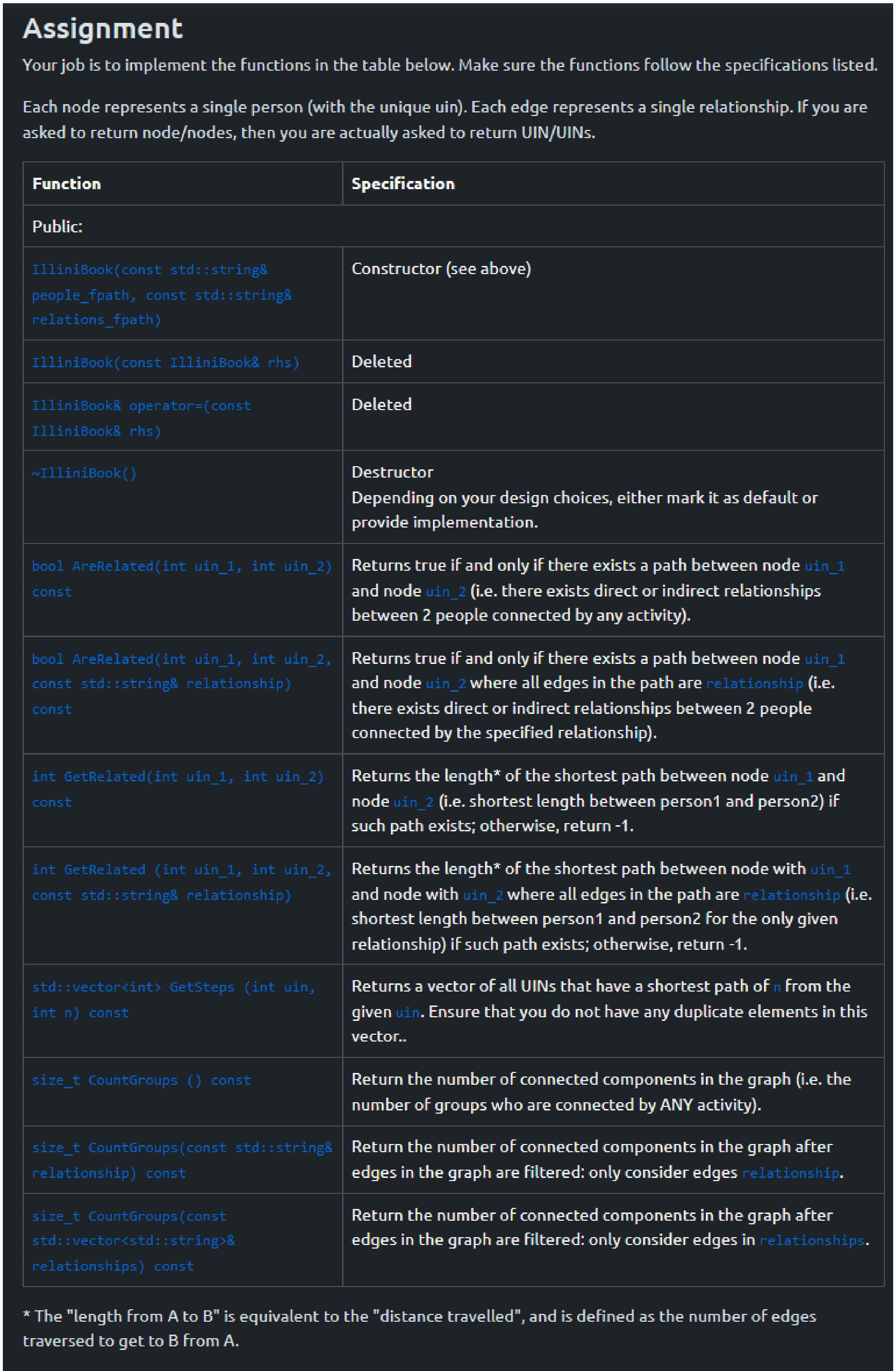
With the paths to these files provided to your constructor, construct an IlliniBook class that implements the functions outlined in the following section. How you store this information is completely up to you.

Assignment

Your job is to implement the functions in the table below. Make sure the functions follow the specifications listed.

Each node represents a single person (with the unique uin). Each edge represents a single relationship. If you are asked to return node/nodes, then you are actually asked to return UIN/UINs.

Function	**Specification**
Public:	
`IlliniBook(const std::string&`	Constructor (see above)



Assignment

Your job is to implement the functions in the table below. Make sure the functions follow the specifications listed.

Each node represents a single person (with the unique uin). Each edge represents a single relationship. If you are asked to return node/nodes, then you are actually asked to return UIN/UINs.

Function	**Specification**
Public:	
`IlliniBook(const std::string& people_fpath, const std::string& relations_fpath)`	Constructor (see above)
`IlliniBook(const IlliniBook& rhs)`	Deleted
`IlliniBook& operator=(const IlliniBook& rhs)`	Deleted
`~IlliniBook()`	Destructor Depending on your design choices, either mark it as default or provide implementation.
`bool AreRelated(int uin_1, int uin_2) const`	Returns true if and only if there exists a path between node `uin_1` and node `uin_2` (i.e. there exists direct or indirect relationships between 2 people connected by any activity).
`bool AreRelated(int uin_1, int uin_2, const std::string& relationship) const`	Returns true if and only if there exists a path between node `uin_1` and node `uin_2` where all edges in the path are `relationship` (i.e. there exists direct or indirect relationships between 2 people connected by the specified relationship).
`int GetRelated(int uin_1, int uin_2) const`	Returns the length* of the shortest path between node `uin_1` and node `uin_2` (i.e. shortest length between person1 and person2) if such path exists; otherwise, return -1.
`int GetRelated (int uin_1, int uin_2, const std::string& relationship)`	Returns the length* of the shortest path between node with `uin_1` and node with `uin_2` where all edges in the path are `relationship` (i.e. shortest length between person1 and person2 for the only given relationship) if such path exists; otherwise, return -1.
`std::vector<int> GetSteps (int uin, int n) const`	Returns a vector of all UINs that have a shortest path of `n` from the given `uin`. Ensure that you do not have any duplicate elements in this vector..
`size_t CountGroups () const`	Return the number of connected components in the graph (i.e. the number of groups who are connected by ANY activity).
`size_t CountGroups(const std::string& relationship) const`	Return the number of connected components in the graph after edges in the graph are filtered: only consider edges `relationship`.
`size_t CountGroups(const std::vector<std::string>& relationships) const`	Return the number of connected components in the graph after edges in the graph are filtered: only consider edges in `relationships`.

* The "length from A to B" is equivalent to the "distance travelled", and is defined as the number of edges traversed to get to B from A.

Assumptions

For the purposes of this MP, you can assume the following about CSVs:

- All UINs in the first CSV are unique
- All pairs of UINs in the second CSV are unique
- All UINs in the second CSV are contained in the first CSV
- Any person will never relationed to themselves.
- There is at most one relationship between two people. (e.g. the second CSV file won't claim that Person A and Person B are connected by both Basketball and CS128.)
- All relationships are symmetric (e.g. if A is related to B by relationship R, then B is also related to A by R.)

You can assume the following about parameters:

- `people_fpath` and `relations_fpath` are valid
- `uin_1 != uin_2`
- `uin`, `uin_1`, `uin_2` are contained in the first csv
- There are no duplicate elements in `relationships`
- `n >= 1`
- `relationships.size() >= 1`

You CANNOT assume the following:

- `relationship` or elements in `relationships` are contained in the second csv (relation csv)
  - If a relationship is not contained in the second csv, you should ignore it.

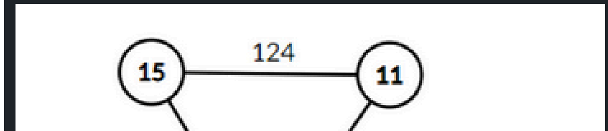
Example

Consider the following input:

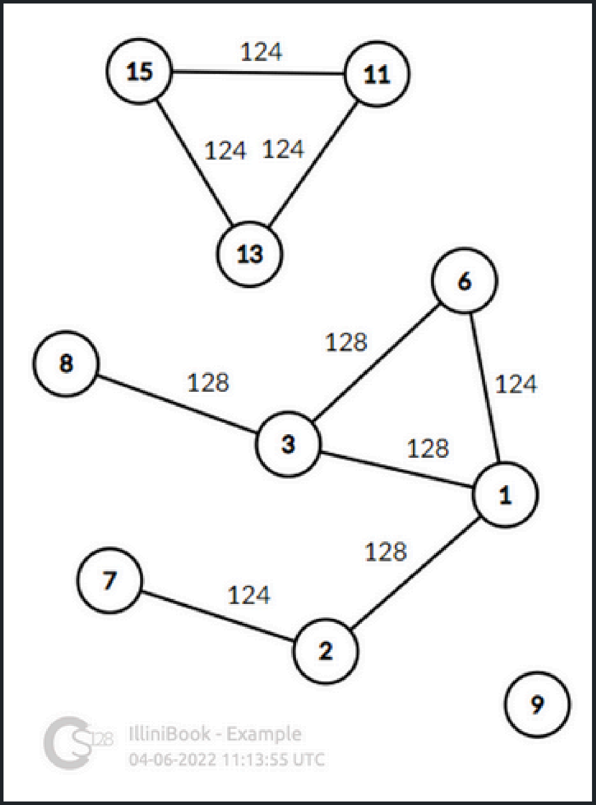
```
persons.csv
1
2
3
6
7
8
9
11
13
15
```

```
relations.csv
1,3,128
3,8,128
3,6,128
1,2,128
2,7,124
1,6,124
11,13,124
13,15,124
11,15,124
```

Here is the corresponding graph where the value on the node is the corresponding UIN and the value on the edge is the corresponding relationship.



Here is the corresponding graph where the value on the node is the corresponding UIN and the value on the edge is the corresponding relationship.



Invocation	Return
`AreRelated(1, 2)`	`true`
`AreRelated(3, 2)`	`true`
`AreRelated(1, 9)`	`false`
`AreRelated(1, 2, "128")`	`true`
`AreRelated(1, 2, "124")`	`false`
`AreRelated(1, 6, "128")`	`true`
`AreRelated(1, 6, "124")`	`true`
`GetRelated(1, 2)`	`1`
`GetRelated(3, 2)`	`2`
`GetRelated(1, 9)`	`-1`
`GetRelated(1, 2, "128")`	`1`
`GetRelated(1, 2, "124")`	`-1`
`GetRelated(1, 6, "128")`	`2`
`GetRelated(1, 6, "124")`	`1`
`GetSteps(1, 1)`	`{ 2, 3, 6 }`*
`GetSteps(1, 2)`	`{ 7, 8 }`*
`GetSteps(1, 3)`	`{ }`
`GetSteps(9, 1)`	`{ }`
`CountGroups()`	`3`

Invocation	Return
`AreRelated(1, 2)`	`true`
`AreRelated(3, 2)`	`true`
`AreRelated(1, 9)`	`false`
`AreRelated(1, 2, "128")`	`true`
`AreRelated(1, 2, "124")`	`false`
`AreRelated(1, 6, "128")`	`true`
`AreRelated(1, 6, "124")`	`true`
`GetRelated(1, 2)`	`1`
`GetRelated(3, 2)`	`2`
`GetRelated(1, 9)`	`-1`
`GetRelated(1, 2, "128")`	`1`
`GetRelated(1, 2, "124")`	`-1`
`GetRelated(1, 6, "128")`	`2`
`GetRelated(1, 6, "124")`	`1`
`GetSteps(1, 1)`	`{ 2, 3, 6 }`*
`GetSteps(1, 2)`	`{ 7, 8 }`*
`GetSteps(1, 3)`	`{ }`
`GetSteps(9, 1)`	`{ }`
`CountGroups()`	`3`
`CountGroups("128")`	`6`
`CountGroups("124")`	`6`
`CountGroups("173")`	`10`
`CountGroups(std::vector<std::string>{ "128", "173" })`	`6`
`CountGroups(std::vector<std::string>{ "128", "124", "173" })`	`3`

**Order does not matter*

Constraints

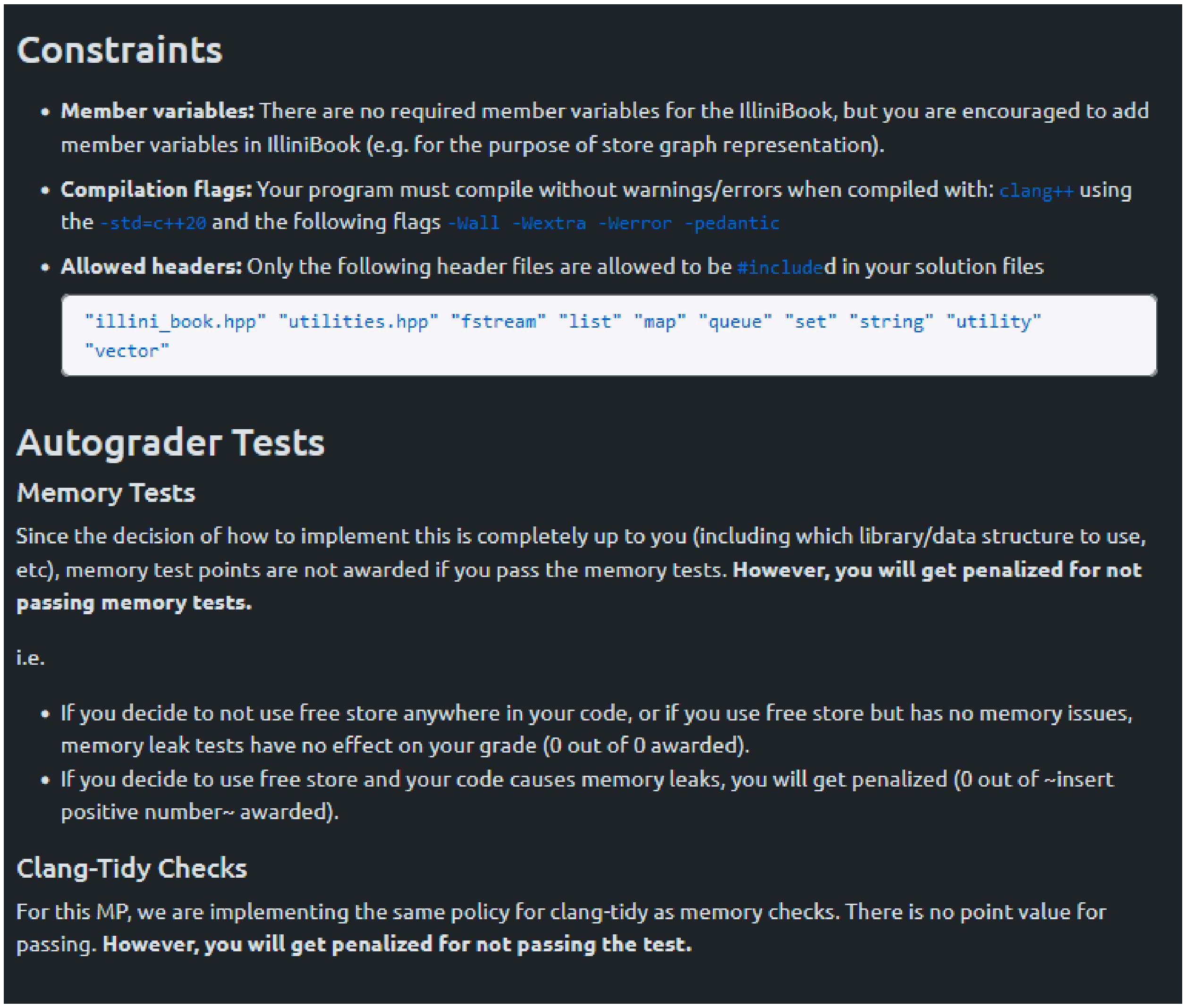
- **Member variables:** There are no required member variables for the IlliniBook, but you are encouraged to add member variables in IlliniBook (e.g. for the purpose of store graph representation).
- **Compilation flags:** Your program must compile without warnings/errors when compiled with: `clang++` using the `-std=c++20` and the following flags `-Wall -Wextra -Werror -pedantic`
- **Allowed headers:** Only the following header files are allowed to be `#include`d in your solution files

  ```
  "illini_book.hpp" "utilities.hpp" "fstream" "list" "map" "queue" "set" "string" "utility"
  "vector"
  ```

Autograder Tests

Memory Tests

Since the decision of how to implement this is completely up to you (including which library/data structure to use, etc), memory test points are not awarded if you pass the memory tests. **However, you will get penalized for not passing memory tests.**



Constraints

- **Member variables:** There are no required member variables for the IlliniBook, but you are encouraged to add member variables in IlliniBook (e.g. for the purpose of store graph representation).
- **Compilation flags:** Your program must compile without warnings/errors when compiled with: `clang++` using the `-std=c++20` and the following flags `-Wall -Wextra -Werror -pedantic`
- **Allowed headers:** Only the following header files are allowed to be `#include`d in your solution files

```
"illini_book.hpp" "utilities.hpp" "fstream" "list" "map" "queue" "set" "string" "utility"
"vector"
```

Autograder Tests

Memory Tests

Since the decision of how to implement this is completely up to you (including which library/data structure to use, etc), memory test points are not awarded if you pass the memory tests. **However, you will get penalized for not passing memory tests.**

i.e.

- If you decide to not use free store anywhere in your code, or if you use free store but has no memory issues, memory leak tests have no effect on your grade (0 out of 0 awarded).
- If you decide to use free store and your code causes memory leaks, you will get penalized (0 out of ~insert positive number~ awarded).

Clang-Tidy Checks

For this MP, we are implementing the same policy for clang-tidy as memory checks. There is no point value for passing. **However, you will get penalized for not passing the test.**