Party problems

Submission deadline:	**2021-11-14 23:59:59**
Late submission with malus:	**2022-02-01 23:59:59** (Late submission malus: 100.0000 %)
Evaluation:	**10.0000**
Max. assessment:	**2.0000** (Without bonus points)
Submissions:	17 / **-**
Advices:	0 / 0

After a brutal loss of territory, the government decided to restrict the political competition to be able to govern more effectively. They reduced the number of political parties to be at most *N*. They also reduced the number of politicians. Moreover, each politician and party must register and obtain a unique registration number.

The current situation affects not only politicians but also journalists. They must keep track of all changes that take place in political parties. And it is no more the year 1938. People want the news in real-time. And not too long, one tweet is enough for everyone.

You discuss the situation in the office and one of your colleagues comes with the following idea. He works in sport redaction and they use a bot to automatically produce news with statistics, match results and so on. What about using the same bot for producing information about political parties. In the long run, it is also statistics.

Your goal is to write a program that helps the editorial office to automatically produce up-to-date news from the political environment.

Bot interface

Your task is to implement the core of a bot that automatically produces newspaper articles. This core part of the bot takes care of information storage. It must contain actual information about political parties, members of political parties and coalitions of parties. The core must have the following interface:

```
class CPulitzer {
    public:

        CPulitzer ( size_t N, size_t P );

        bool register_politician ( uint32_t id_party, uint32_t id_politician, const string &
  name, uint32_t popularity, uint8_t gender );

        bool politician_name ( uint32_t id_politician, string & name ) const;

        bool politician_gender ( uint32_t id_politician, uint8_t & gender ) const;

        bool politician_popularity ( uint32_t id_politician, uint32_t & popularity ) const;

        bool deregister_politician ( uint32_t id_politician );

        bool party_leader ( uint32_t id_party, uint32_t & id_leader ) const;

        bool change_popularity ( uint32_t id_politician, uint32_t popularity );

        bool sack_leader ( uint32_t id_party );

        bool merge_parties ( uint32_t dest_party, uint32_t src_party );

        bool create_coalition ( uint32_t id_party1, uint32_t id_party2 );

        bool leave_coalition ( uint32_t id_party );

        bool coalition_leader ( uint32_t id_party, uint32_t & id_leader ) const;

        bool scandal_occured ( uint32_t id_party );
```

```
};
```

`CPulitzer::CPulitzer`
: Bot implementation constructor. The parameters are the number of allowed political parties and the number of allowed politicians. It holds that $1 \leq N, P \leq 10^6$;

`CPulitzer::register_politician`
: The method registers a new politician and add him to the political party. If the specified politician id is already in use, the method returns `false`. Otherwise, it returns `true`.

`Cpulitzer::deregister_politician`
: The method deregisters existing politician. If no politician with a given id is registered, then the method returns `false`. Otherwise, the politician is deregistered, his id is returned for future use, and the method returns `true`.

`CPulitzer::party_leader`
: The method determines the leader of the given party. If the party does not exist, the method returns `false`. Otherwise, the output parameter `id_leader` contains the id of the politician with the highest popularity through all party members. If two politicians have the same popularity, then the leader is the politician who received this popularity at a later time.

`CPulitzer::change_popularity`
: The method updates politicians' popularity. If no politician with a given id is registered, then the method returns `false`. Otherwise, politicians' popularity is changed and the method returns `true`.

`CPulitzer::sack_leader`
: Due to the results of the last elections, the leader of the given party leaves politics. The method returns `false` if there is no party with the given id, otherwise, it returns `true`.

`CPulitzer::merge_parties`
: The method merges `dest_party` with `src_party`. All members of `src_party` are trasfered into `dest_party` and the `src_party` is canceled. If both parties exists, the method returns `true`, otherwise, it returns `false`.

`CPulitzer::create_coalition`
: Two parties decided to create a coalition. If any of the parties is already part of the coalition, then all coalitions are merged. If the party with the given id does not exist, the method returns `false`. Otherwise, it returns `true`.

`CPulitzer::leave_coalition`
: The given party is leaving the coalition. If the party is not a member of a coalition, or the party does not exist at all, the method returns `false`. Otherwise, the method returns `true`.

`CPulitzer::scandal_occured`
: A scandal broke out. The leader of the given political party should leave politics. If the party is a member of a coalition, the same holds for the coalition leader. If the party does not exist, the method returns `false`. Otherwise, the method returns `true`.

`CPulitzer::coalition_leader`
: The method obtains the current coalition leader, which is a politician with the highest popularity over all parties in the coalition. If the specified party is not part of any coalition, then the method outputs the party's leader and returns `true`. If the party with a specified id does not exist, the method returns `false`.

- Methods `politician_name`, `politician_popularity`, and `politician_gender` retrieves personal information about the specified politician. All the methods return `false` if there is no registered politician with the given id. Otherwise, information is stored in an output parameter and methods return `true`
- Party ID is always an integer from interval $[0; N - 1]$.
- Politician ID is always an integer from interval $[0; P - 1]$.
- If also the last member of a party leaves, the party is cancelled. Its ID is then free for future use.

Examples

Example 1

```
uint8_t gender;
uint32_t popularity, id_leader;
std::string name;

CPulitzer bot( 3, 10 );
bot.party_leader( 1, id_leader ); // false
bot.register_politician( 1, 5, "VK", 1000, 77 ); // true
bot.register_politician( 2, 4, "MZ", 1000, 77 ); // true
bot.register_politician( 2, 7, "VS", 500, 77 ); // true
bot.party_leader( 1, id_leader ); // true, 5
bot.party_leader( 2, id_leader ); // true, 4
bot.change_popularity( 7, 2000 ); // true
bot.party_leader( 2, id_leader ); // true, 7
bot.register_politician( 1, 2, "MT", 500, 77 ); // true
bot.register_politician( 2, 2, "JP", 500, 77 ); // false
```

```
bot.register_politician( 2, 9, "JP", 500, 77 ); // true
bot.deregister_politician( 5 ); // true
bot.party_leader( 1, id_leader ); // true, 2
bot.sack_leader( 2 ); // true
bot.change_popularity( 9, 200 ); // true
bot.sack_leader( 1 ); // true
```

Example 2

```
uint8_t gender;
uint32_t popularity, id_leader;
std::string name;

CPulitzer bot( 5, 5 );
bot.register_politician( 0, 0, "RS", 150, 77 ); // true
bot.register_politician( 1, 1, "RS", 50, 77 ); // true
bot.register_politician( 2, 2, "RS", 60, 77 ); // true
bot.register_politician( 3, 3, "VKml", 100, 77 ); // true
bot.register_politician( 3, 4, "ZMZ", 50, 70 ); // true
bot.deregister_politician( 3 ); // true
bot.merge_parties( 3, 2 ); // true
bot.merge_parties( 3, 1 ); // true
bot.party_leader( 0, id_leader ); // true, 0
bot.party_leader( 1, id_leader ); // false
bot.party_leader( 2, id_leader ); // false
bot.party_leader( 3, id_leader ); // true, 2
```

Example 3

```
uint8_t gender;
uint32_t popularity, id_leader;
std::string name;

CPulitzer bot( 10, 10 );
bot.register_politician( 9, 1, "MK", 100, 77 ); // true
bot.register_politician( 0, 0, "IB", 150, 77 ); // true
bot.register_politician( 1, 2, "VR", 50, 77 ); // true
bot.create_coalition( 9, 1 ); // true
bot.leave_coalition( 1 ); // true
bot.create_coalition( 0, 1 ); // true
bot.coalition_leader( 0, id_leader ); // true, 0
bot.coalition_leader( 1, id_leader ); // true, 0
bot.coalition_leader( 9, id_leader ); // true, 1
bot.change_popularity( 2, 200 ); // true
bot.coalition_leader( 0, id_leader ); // true, 2
bot.leave_coalition( 9 ); // false
```

Classification conditions

- To obtain 2 points, you must correctly solve instances without `merge_parties` and `create_coalition` methods calls.
- To obtain 5 points, you must correctly solve instances without `create_coalition` methods calls.
- To obtain 10 points, you must correctly solve all unrestricted instances.

Notes

- We recommend reading the assignment twice before coding. It can help you to choose an appropriate data structure, which is optimally extendable.
- The STL library is not allowed.
- Automated journalism is another big think. If interested, we recommend *Automated Jurnalism* article on **Wikipedia** as good starting point.

Submit: Choose File **Submit**