Cruise

Let's define two structures:

```
struct Passenger
{
    string name, surname;
    Passenger * pNext;
    // some other data
};

struct Cruise
{
    string start_harbour, end_harbour;
    string date;
    Passenger * pPassengers;
    Cruise * pPrev, * pNext;
    // some other data
};
```

The structures are used to build a complicated data structure presented in Fig. 3. Cruises build a doubly link list in non descending order by dates. Passengers in a cruise are gathered in a singly linked list in non descending order by their surnames. A cruise may have any length of passengers (also no passengers).

Define functions:

1. `Cruise * findCruise (Cruise * pH, const string & start_harbour, const string & end_harbour, const string & date);`

   The function searches for a cruise defined with a date, start harbour, and end harbour in a list of cruises `pH`. The function returns an address of the found cruise. If there is not matching cruise, the function returns `NULL` (or **nullptr** or 0).

2. `Cruise * addCruise (Cruise * & pH, Cruise * & pTail, const string & date, const string & start_harbour, const string & end_harbour);`

   The function adds in a non descending order a new cruise to a list of cruises (head: `pH`, tail `pT`). The adding of a new cruise keeps the non descending order by date. The function returns an address of an added cruise. It is possible that the list is empty. The function add a new cruise **iteratively**.

3. `Passenger * addPassengerToCruise (Passenger * pH, const string & name, const string & surname);`

   The function adds a new passenger to a singly linked list of cruises (head `pH`). The function returns a head of a modified list. The function keeps the non descending order by surnames of passengers. An empty list is possible. The function adds passenger **recursively**.

4. `void addPassenger (Cruise * & pH, Cruise * & pT, const string & date, const string & start_harbour, const string & end_harbour, const string & name, const string & surname)`

   The function add a passenger (name, surname given) to a doubly linked list of cruises (with head `pH` and tail `pT`). If a cruise is missing, it is added so that the ordering of the list is kept. It is possible that the list of cruised is empty. Cruises with no passengers are also possible. Function uses the functions defined above.

5. `Cruise * favourite (Cruise * & pH)`

   The function returns an address of a cruise with the maximal length of passengers. Passed parameters: an address of the head of the list `pH`. If the list of cruises is empty, the function returns

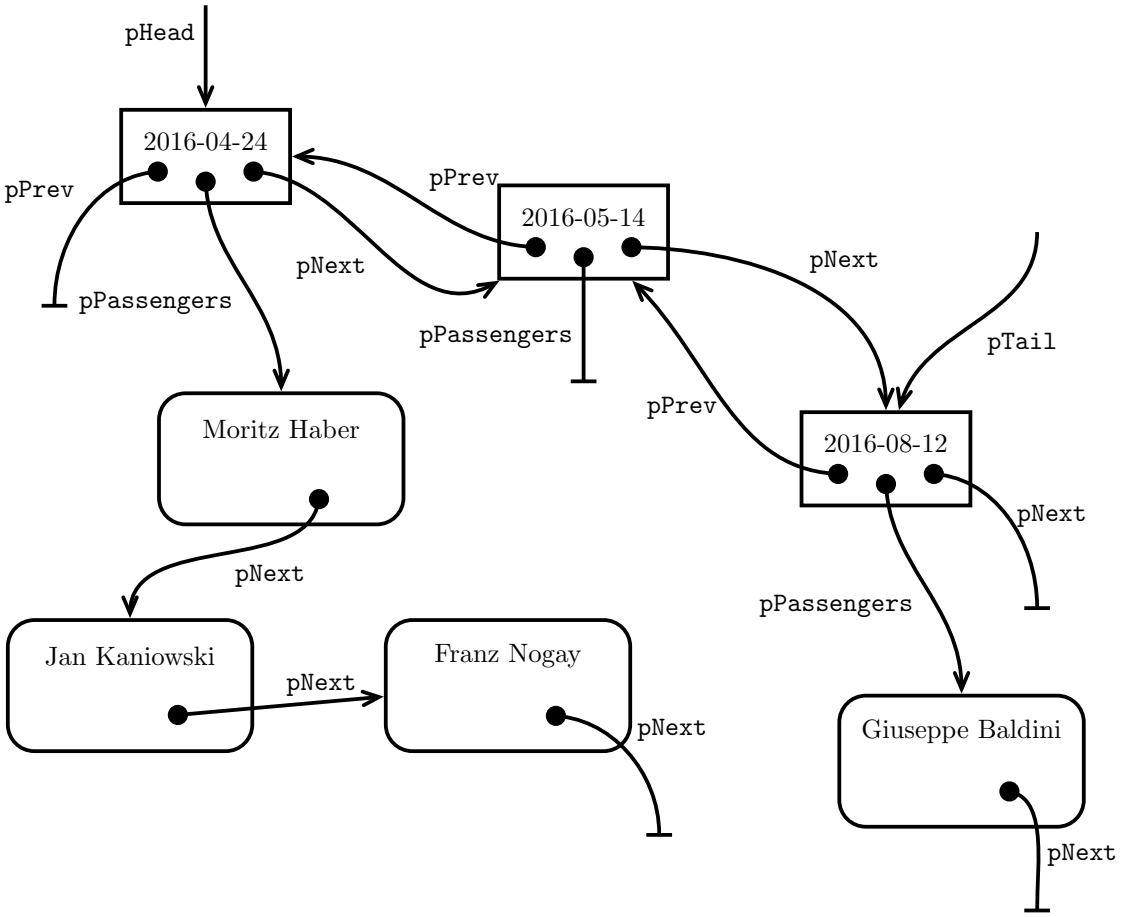

Figure 3: Example of cruises and passengers.

`NULL` (or **nullptr** or 0). If several cruises have the same lengths of passengers, the function returns an address of any of cruises with maximal length of passengers.

Attention: You may not modify the definitions of `Cruise` and **Passenger** structures.