# CS 201, Fall 2025

Homework Assignment 3

Due: 23:59, December 12, 2025

## 1 Introduction

In this homework, you will implement a system called **BilkentTourism**, which manages bus lines, stops, and buses using linked lists. Each bus line, stop, and bus is represented by a unique ID, and these ID types are independent from one another; for example, a stop and a bus may share the same numeric ID but still represent different entities. For example, there cannot be more than one bus with ID 2, but there can be a bus line with ID 2 and stop with ID 2. They would be different entities.

Your implementation should maintain sorted linked lists of bus lines, stops and buses. All of these structures must be implemented strictly using linked lists as specified.

The `BilkentTourism` class you will implement must support adding, removing, and printing bus lines, stops, and buses, fully adhering to the required behaviors and output formats. In your implementation, you MUST use **sorted linked lists** as the main data structure for storing all entities.

### 1.1 Bus Lines

Each bus line has a unique ID, a name, a sorted list of stops and a sorted list of buses. You need to use a sorted linked list to store the bus lines with respect to ascending line IDs. The required functions are as follows:

- `addBusLine`: Adds a new bus line to the system with an ID and a name. Since IDs must be unique, the system must check whether or not the specified bus line ID already exists, and if it exists, it must not allow the operation and display a warning message. Initially, the newly added bus line does not have any bus stops or buses. Example log messages:
  - Added bus line 6.
  - Cannot add line. There is already a bus line with ID 6.
- `removeBusLine`: The function removes the bus line with the specified line ID. If the bus line does not exist, the system must display a warning message. If it exists, the entire line must be deleted along with all buses assigned to it. Example log messages:
  - Removed bus line 17.
  - Cannot remove bus line. There is no bus line with ID 21.

- `printBusLines`: Prints IDs and names of all bus lines in the system. If there is no bus line in the system, give a warning message. The entries must be shown in ASCENDING ORDER with respect to their IDs. Example log messages:
  - There are no bus lines to show.
  - Bus lines in the system:
    Line 11 : KM18
    Line 23 : TK10
    Line 25 : PG26
    Line 32 : MY08

### 1.2 Stops

Each bus line stores a list of stops. These stops must be maintained in a sorted linked list in ascending order by stop ID. Each stop is identified by a unique ID. The required functions are as follows:

- `addStop`: Adds new stop to the system with an ID and a name. Since IDs must be unique, the system must check whether or not the specified stop ID already exists, and if it exists, it must not allow the operation and display a warning message. Example log messages:
  - Added stop 25.
  - Cannot add stop. BilkentTourism already contains stop 6.
- `removeStop`: Removes the specified stop from the system. If the stop does not exist or is still used by other lines, the system must display a warning message. Example log messages:
  - Removed stop 17.
  - Cannot remove stop 17. There is no bus stop with ID 17.
  - Cannot remove stop 17. The stop is currently in use.
- `addStopToLine`: Adds a stop to the specified bus line. The system must first check whether the stop exists, and then check whether the bus line exists. If either the bus line or the stop does not exist, the operation must be rejected and a warning message must be displayed. If both the stop and the bus line exist, the system must check whether the stop is already present in that line. If the stop already exists in the line, a warning message must be shown and the operation must not proceed. Otherwise, the stop must be inserted into the line's stop list in ascending order by stop ID. Note that the same stop may be added to multiple lines.

  Example log messages:
  - Added stop 48 to line 20 (T10).

  - Cannot add stop. There is no stop with ID 48.
  - Cannot add stop. There is no line with ID 20.
  - Cannot add stop. Line 20 already contains stop 48.

- `removeStopFromLine`: Removes the stop from the given bus line. If the bus line does not exist, give a warning message. If the bus line exists but does not contain the given stop, the system must also print a warning message. If the stop exists, it must be removed from the stop list of that line. Example log messages:
  - Removed stop 48 from line 5.
  - Cannot remove stop. Line 5 does not have stop 48.
  - Cannot remove stop. There is no line with ID 20.

- `printStops`: Prints the IDs, and names of all stops in the bus line. If there is no stop in the bus line, give a warning message. The entries should be shown in ASCENDING ORDER according to IDs. Example log messages:
  - Cannot print stops. There is no line with ID 20.
  - Cannot print stops. There is no stop to show.
  - Stops in bus line 4 (59RK) :
    Stop 14 : Tunus
    Stop 26 : Incek
    Stop 33 : Bahceli
    Stop 48 : Umitkoy

### 1.3 Busses

Each bus is identified by a unique ID, which you can think of as its license plate number. Each bus is assigned to a single bus line, and a bus can belong to only one line at a time. The required functions are as follows:

- `assignBus`: Assign a new bus to the line with an ID. The system must first check whether the bus is assigned to a line. If there is already a bus with the same ID in a different line, it must not allow the operation and display a warning message. Example log messages:
  - Bus 6 with driver David Bowie is assigned to line 5 (T10).
  - Cannot assign bus. There is no line with ID 2.
  - Cannot assign bus. Bus 6 is already assigned to a line.

- `unassignBus`: The system will allow unassigning bus from line. If the bus does not exist, the system must display a warning message. Example log messages:
  - Bus 7 is unassigned from line 5 (T10).

  - Cannot unassign bus. There is no bus with ID 7.

- `printBussesInLine`: Prints the IDs, and driver names of the buses assigned to the specified line. If the line does not exist or no buses are assigned to it, a warning message should be displayed. The bus IDs must be shown in ascending order. Example log messages:

  - Cannot print busses. There is no line with ID 2.
  - There are no bus to show in line 5 (T10).
  - Busses and their drivers assigned to the line 5 (T10):
    Bus 14 : Michael Scott
    Bus 45 : Jim Halpert
    Bus 68 : Creed Bratton

- `printBussesPassingStop`: Prints the IDs of all lines passing through the specified stop, along with the IDs of the buses assigned to those lines. If the stop does not exist or no bus uses that stop, a warning message should be displayed. Bus IDs must be listed in ascending order. Example log messages:

  - Cannot print buses. There is no stop with ID 4.
  - Cannot print buses. No lines pass through the stop.
  - Buses and their assigned lines passing the stop 4 (Tunus):
    Line 14 (59RK) : [23, 65, 87]
    Line 45 (T10) : [3, 6, 55]
    Line 63 (H85) : None
    Line 74 (K10) : [9]

Below is the required public part of the `BilkentTourism` class that you must write in this assignment. The name of the class must be `BilkentTourism`, and must include these public member functions. The interface for the class must be written in the file called `BilkentTourism.h` and its implementation must be written in the file called `BilkentTourism.cpp`. You can define additional public and private member functions and data members in this class. You can also define additional classes in your solution and implement them in separate files.

```
class BilkentTourism {
public:
    BilkentTourism();
    ~BilkentTourism();

    void addBusLine( const int lineId, const string lineName );
    void removeBusLine( const int lineId );
    void printBusLines() const;

    void addStop( const int stopId, const string stopName  );
    void removeStop( const int stopId );
    void addStopToLine( const int stopId, const int lineId );
    void removeStopFromLine( const int stopId, const int lineId );
    void printStops( const int lineId ) const;

    void assignBus( const int busId, const string driverName, const int lineId);
    void unassignBus( const int busId );
    void printBussesInLine( const int lineId ) const;
    void printBussesPassingStop( const int stopId ) const;

};
```

Here is an example test program that uses this class and the corresponding output. We will use a similar program to test your solution so make sure that the name of the class is `BilkentTourism`, its interface is in the file called `BilkentTourism.h`, and the required functions are defined as shown above. Your implementation **MUST** use exactly the same format given in the example output to display the messages expected as the result of the defined functions.

**Example test code:**

```cpp
#include <iostream>
using namespace std;
#include "BilkentTourism.h"

int main() {
    BilkentTourism bilkentTourism;
    bilkentTourism.printBusLines(); // No bus to show
    cout<<endl;
    bilkentTourism.addBusLine(25, "T10");
    bilkentTourism.addBusLine(22, "G14");
    bilkentTourism.addBusLine(25, "C16"); // Cannot add. Already added
    bilkentTourism.addBusLine(28, "C16");
    bilkentTourism.addBusLine(26, "T43");
    cout<<endl;
    bilkentTourism.removeBusLine(15); // //Cannot remove. No Line.
    bilkentTourism.removeBusLine(26);
    cout<<endl;
    bilkentTourism.printBusLines();
    cout<<endl;
    bilkentTourism.addStop(4,"Besevler");
    bilkentTourism.addStop(7,"Tunali");
    bilkentTourism.addStop(3,"Kizilay");
    bilkentTourism.addStop(9,"Batikent");
    bilkentTourism.addStop(6,"Ovecler");
    bilkentTourism.addStop(1,"Bahceli");
    bilkentTourism.addStop(6,"Ovecler"); //Cannot add. Already Added
    cout<<endl;
    bilkentTourism.addStopToLine(7,25);
    bilkentTourism.addStopToLine(3,25);
    bilkentTourism.addStopToLine(99,25); // Cannot add stop. No stop
    bilkentTourism.addStopToLine(6,99); // Cannot add stop. No line
    bilkentTourism.addStopToLine(3,25); // Cannot add stop. Already has stop
    bilkentTourism.addStopToLine(6,25);
    bilkentTourism.addStopToLine(3,22);
    bilkentTourism.addStopToLine(1,22);
    bilkentTourism.addStopToLine(7,28);
    cout<<endl;
    bilkentTourism.removeStop(4);
    bilkentTourism.removeStop(99); // Cannot remove. No Stop
    bilkentTourism.removeStop(3);  //Cannot remove. Stop in use
    cout<<endl;
    bilkentTourism.removeStopFromLine(7,99); //Cannot remove. No line
    bilkentTourism.removeStopFromLine(7,22); //Cannot remove. No stop in line
    bilkentTourism.removeStopFromLine(7,28);
```

```
    cout<<endl;
    bilkentTourism.printStops(99); //Cannot print. No line
    bilkentTourism.printStops(28); //Cannot print. No stops
    bilkentTourism.printStops(25);
    cout<<endl;
    bilkentTourism.assignBus(32,"David",25);
    bilkentTourism.assignBus(38,"Kemal",25);
    bilkentTourism.assignBus(33,"Ayse",22);
    bilkentTourism.assignBus(35,"Hasan",22);
    bilkentTourism.assignBus(33,"Ayse",99); //Cannot assign. No line
    bilkentTourism.assignBus(32,"David",22); //Cannot assign. Already assigned
    cout<<endl;
    bilkentTourism.unassignBus(35);
    bilkentTourism.unassignBus(99); //Cannot unassign. No bus
    cout<<endl;
    bilkentTourism.printBussesInLine(28);//Cannot print. No bus
    bilkentTourism.printBussesInLine(99);//Cannot print. No line
    bilkentTourism.printBussesInLine(25);
    cout<<endl;
    bilkentTourism.printBussesPassingStop(99); //Cannot print. No line
    bilkentTourism.printBussesPassingStop(9); //Cannot print. No lines pass through
    bilkentTourism.printBussesPassingStop(3);


}
```

**Output of the example test code:**

```
There are no bus lines to show.

Added bus line 25.
Added bus line 22.
Cannot add line. There is already a bus line with ID 25.
Added bus line 28.
Added bus line 26.

Cannot remove bus line. There is no bus line with ID 15.
Removed bus line 26.

Bus lines in the system:
Line 22 : G14
Line 25 : T10
Line 28 : C16

Added stop 4.
Added stop 7.
```

```
Added stop 3.
Added stop 9.
Added stop 6.
Added stop 1.
Cannot add stop. BilkentTourism already contains stop 6.

Added stop 7 to line 25 (T10).
Added stop 3 to line 25 (T10).
Cannot add stop. There is no stop with ID 99.
Cannot add stop. There is no line with ID 99.
Cannot add stop. Line 25 already contains stop 3.
Added stop 6 to line 25 (T10).
Added stop 3 to line 22 (G14).
Added stop 1 to line 22 (G14).
Added stop 7 to line 28 (C16).

Removed stop 4.
Cannot remove stop 99. There is no bus stop with ID 99.
Cannot remove stop 3. The stop is currently in use.

Cannot remove stop. There is no line with ID 99.
Cannot remove stop. Line 22 does not have stop 7.
Removed stop 7 from line 28.

Cannot print stops. There is no line with ID 99.
Cannot print stops. There is no stop to show.
Stops in bus line 25 (T10) :
Stop 3 : Kizilay
Stop 6 : Ovecler
Stop 7 : Tunali

Bus 32 with driver David is assigned to line 25 (T10).
Bus 38 with driver Kemal is assigned to line 25 (T10).
Bus 33 with driver Ayse is assigned to line 22 (G14).
Bus 35 with driver Hasan is assigned to line 22 (G14).
Cannot assign bus. There is no line with ID 99.
Cannot assign bus. Bus 32 is already assigned to a line.

Bus 35 is unassigned from line 22 (G14)
Cannot unassign bus. There is no bus with ID 99

There are no bus to show in line 28 (C16).
Cannot print busses. There is no line with ID 99.
Busses and their drivers assigned to the line 25 (T10):
Bus 32 : David
Bus 38 : Kemal
```

```
65
66 Cannot print buses. There is no stop with ID 99.
67 Cannot print buses. No lines pass through the stop.
68 Buses and their assigned lines passing the stop 3 (Kizilay):
69 Line 22 (G14) : [33]
70 Line 25 (T10) : [32,38]
```

## 2 Specifications

1. You ARE NOT ALLOWED to modify the given parts of the header file. You MUST use **sorted linked lists** in your implementation. You will get no points if you use dynamic or fixed-sized arrays or any other data structures such as vectors/arrays/lists from the standard library. However, if necessary, you may define additional data members and member functions. You may also define additional classes.

2. Moreover, you ARE NOT ALLOWED to use any global variables or any global functions.

3. Output message for each operation MUST match the format shown in the output of the example code. Your output will be compared verbatim with the expected output during evaluation.

4. Your code MUST NOT have any memory leaks. You will lose points if you have memory leaks in your program even though the outputs of the operations are correct. To detect memory leaks, you may want to use Valgrind which is available at http://valgrind.org.

## 3 Submission

1. In this assignment, you must have separate interface and implementation files (i.e., separate `.h` and `.cpp` files) for your class. Your class name MUST BE `BilkentTourism` and your file names MUST BE `BilkentTourism.h` and `BilkentTourism.cpp`. Note that you may write additional class(es) in your solution.

2. The code (`main` function) given above is just an example. We will test your implementation using different scenarios, which will contain different function calls. Thus, do not test your implementation only by using this example code. We recommend you to write your own driver files to make extra tests. However, you MUST NOT submit these test codes (we will use our own test code). In other words, do not submit a file that contains a function called `main`.

3. You should put all of your `.h` and `.cpp` files into a folder and zip the folder (in this zip file, there should not be any file containing a `main` function). The name of this zip file should conform to the following name convention: secX-Firstname-Lastname-StudentID.zip where X is your section number. The submissions that do not obey these rules will not be graded.

4. Make sure that each file that you submit (each and every file in the archive) contains your name, section, and student number at the top as comments.

5. You are free to write your programs in any environment (you may use Linux, Windows, MacOS, etc.). On the other hand, we will test your programs on "dijkstra.ug.bcc.bilkent.edu.tr" and we will expect your programs to compile and run on the dijkstra machine. Your code will be tested by using an automated test suite that includes multiple test cases where each case corresponds to a specific number of points in the overall grade. We will provide you with example test cases by email. Thus, we strongly recommend you to make sure that your program successfully compiles and correctly works on dijkstra.ug.bcc.bilkent.edu.tr before submitting your assignment. If your current code does not fully compile on dijkstra before submission, you can try to comment out the faulty parts so that the remaining code can be compiled and tested during evaluation.

6. This assignment is due by 23:59 on Friday, December 12, 2025. You should upload your work to Moodle before the deadline. No hardcopy submission is needed. Late submissions will not be accepted (if you can upload to Moodle, then you are fine). There will be no extension to this deadline.

7. We use an automated tool as well as manual inspection to check your submissions against plagiarism. For questions regarding academic integrity and use of external tools (including generative AI tools), please refer to the course home page and the Honor Code for Introductory Programming Courses (CS 101/102/201/202) at `https://docs.google.com/document/d/1v_3ltpV_lClLsROXrMbojyuv4KrFQAm1uoz3SdC-7es/edit?usp=sharing`.

8. This homework will be graded by your TAs **Pınar Gül** (pinar.gul@bilkent.edu.tr), and **Mehmet Alper Yilmaz** (mehmet.yilmaz@bilkent.edu.tr). Thus, you may ask your homework related questions directly to them. There will also be a forum on Moodle for questions.