In this exercise we implement more overloaded operators and use stringstreams.

Excercise A (15 p) Implement more operators and a new constructor

Improve class you wrote in exercise 4&5 by adding overloaded operators and by splitting it to header and source files.
The operators to add are:

1. Input operator (>>) that extracts time from a stream. The format of the time in the stream is assumed to be same as output (two numbers separated by a colon).
2. Comparison operator greater than (>) that compares two times
3. Comparison operator equal to (==) that compares two times
4. Implement a constructor that takes hours and minutes as parameter

Header should only contain class definition and possibly some includes required by the definition. In the example below `#pragma once` is nowadays widely supported mechanism that works the same way as header guards after the pragma. The header guards are used to prevent including same header multiple times. Note that header should not contain any code that can be executed i.e. no function implementations.

```
#pragma once
#ifndef TIME_H_INCLUDED
#define TIME_H_INCLUDED
#include <iostream>

class Time {
	friend std::ostream& operator<<(std::ostream& out, const Time& s);
	friend std::istream& operator>>(std::istream& in, Time& s);
public:
	void read(const char* s);
	bool lessThan(const Time &t) const;
	Time subtract(const Time &t) const;
	void display(void) const;
	bool operator<(const Time& t) const;
private:
	int hour;
	int min;
};

#endif
```

The source file includes library headers that are needed for the implementation of functions and also includes the header of the class it is implementing. A common practice is to name the files by the class they are implementing. For example, class Time is defined and implemented in Time.h and Time.cpp.

```
#include <iostream>
#include <iomanip>
#include "Time.h"
using namespace std;
void Time::read(const char* s) {
	cout << s << " ";
	cin >> hour >> min;
}
```

Exercise B (15p) String streams

Implement class Day to hold calendar events of one day. The class should have at least the following members:

```
class Day {
public:
    Day();
    bool from_str(const string &s);
    string to_str();
    void dst(int offset);
private:
    int day;
    string month;
    vector<Time> list;
};
```

Member list is used to store times of event on a day. Note that we don't store event names only starting time of each event.
Member function from_str reads events from a string. All existing data (date and event times) are erased by from_str before it makes an attempt to read data. The function returns true if date and at least one event time was successfully read otherwise the function returns false. Function sorts the event times in ascending order before returning.
The string from which to read contains day and month separated by spaces and a list of events start times separated by spaces.

For example:

```
13 January 12:00 14:45
14 January 09:00 11:30 13:15 16:00 12:00
1 February 19:00 21:30
2 February 12:00 14:45
1 April 10:00 11: Broken
1 May 8:00 17:55
2 May 00:01 22:00 13:05
7 May Broken
28 May 10:00 17:45
```

In the example above there is one completely incorrect line (May 7th) that should be completely ignored and one partially incorrect line (April 1st) where the last time is invalid and should be ignored.
Member function to_str returns a string of event times. The format of the string is the same as from_str input string format: day and month separated by spaces followed by a list of event times separated by spaces.
Member function dst adds an offset to times. The offset is number of hours to add. Note that the function needs to roll time properly over 24:00 but **it does not need to change the date on roll over**.
Test your class with the following code and with different data files not just the one example provided in the assignment.

```
int main()
{
    ifstream inputFile("calendar.txt");

    if (!inputFile.is_open()) {
        cout << "Unable to open file" << endl;
        return 1;
    }

    string line;
    vector<Day> cal;
    Day day;

    while (getline(inputFile, line)) {
        if (day.from_str(line)) {
            cal.push_back(day);
        }
    }
    cout << "Calendar" << endl;

    for (auto& e : cal) {
        cout << e.to_str() << endl;
    }

    // DST shift
    for (auto& e : cal) {
        e.dst(1);
    }
    cout << "DST" << endl;

    for (auto& e : cal) {
        cout << e.to_str() << endl;
    }

    cout << "End" << endl;

    return 0;
}
```

Example test output:

```
Calendar
13 January 12:00 14:45
14 January 09:00 11:30 12:00 13:15 16:00
1 February 19:00 21:30
2 February 12:00 14:45
1 April 10:00
1 May 08:00 17:55
2 May 00:01 13:05 22:00
28 May 10:00 17:45
DST
13 January 13:00 15:45
14 January 10:00 12:30 13:00 14:15 17:00
1 February 20:00 22:30
2 February 13:00 15:45
1 April 11:00
1 May 09:00 18:55
2 May 01:01 14:05 23:00
28 May 11:00 18:45
End
```