

Exercise 8a. (STL Algorithms, 0.5p)

Create a program, which has a list of times¹ (using your old class Time). First enter times to that list from the keyboard in a random order. Then order those time entries in a standard way (later time is “bigger”, e.g. 11:03 is bigger than 10:54), and print that ordered list of times.

After that (in the same program) calculate the difference between successive times and print out the same array of times and their time differences like

```
Enter times (terminate with 0 0):? 12 43 8 22 17 45 2 20 0 0
Unsorted times:
12:43
08:22
17:45
02:20

Sorted times:
02:20
08:22
12:43
17:45

Time differences:
02:20
08:22 06:02
12:43 04:21
17:45 05:02
```

Note: use the standard `sort()` and `adjacent_difference()` algorithms for the sorting operation (you may also need `back_inserter()`).

Exercise 8b. (Doubly linked list, 0,5p)

Implement an ADT doubly linked list of integers. The typename of that ADT is `Tdbl`. The following operation functions are needed:

```
constructor      (initializes a doubly linked list)
insert_to_front  (adds a new node to the front)
insert_to_back   (adds a new node to the end)
print            (displays all elements from first to last)
print_reverse    (displays elements from last to first)
```

You can use a simple program below to test your function implementations:

```
int main(void) {
    LinkedList<int> dbl;
```

¹ You can use STL vector for the list

```
    dbl.print();
    dbl.print_reverse();

    dbl.insert_to_back(10);
    dbl.print();
    dbl.print_reverse();

    dbl.insert_to_front(20);
    dbl.print();
    dbl.print_reverse();

    dbl.insert_to_back(30);
    dbl.print();
    dbl.print_reverse();

    dbl.insert_to_front(40);
    dbl.print();
    dbl.print_reverse();

    dbl.insert_to_back(50);
    dbl.print();
    dbl.print_reverse();

    return EXIT_SUCCESS;
}
```

Output from the program should be like this

```
List:
List:
List: 10
List: 10
List: 20 10
List: 10 20
List: 20 10 30
List: 30 10 20
List: 40 20 10 30
List: 30 10 20 40
List: 40 20 10 30 50
List: 50 30 10 20 40
```

Exercise 8c. (Sparse matrix using linked lists, voluntary, 0.5p)

Warning! This is not an easy one (maybe the most difficult exercise in this course).

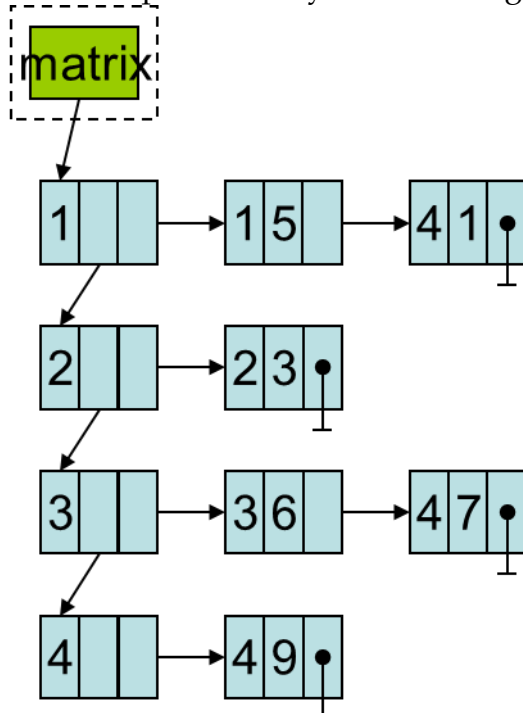
In this exercise sparse matrix using universal item (see exercise 6b) will be created. Sparse matrix is a matrix where most of the cells are empty. Therefore it is efficient to implement the matrix using linked lists. If we have the following matrix

```
5 0 0 1
0 3 0 0
```

0 0 6 7

0 0 0 9

it can be represented by the following two-dimensional linked list structure



where the first column of nodes is a linked list that holds the heads of the linked lists of columns.

Test your solution with the following application

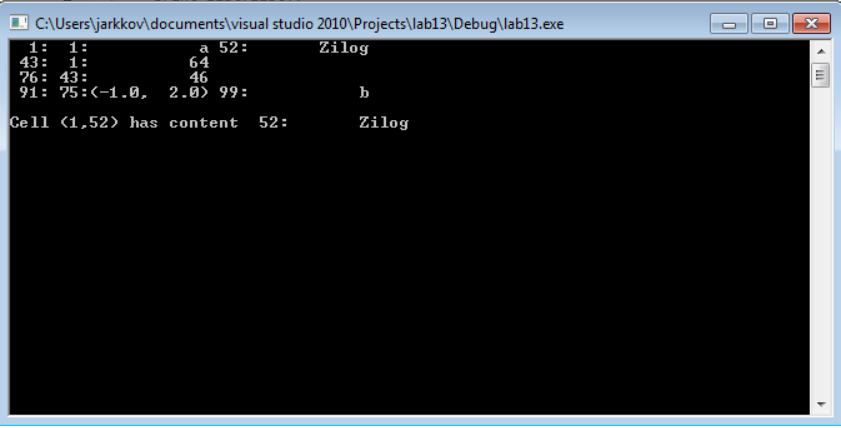
```
int main() {
    Matrix matrix;

    matrix[1][1] = new Char('a');
    matrix[1][52] = new String("Zilog");
    matrix[43][1] = new Int(64);
    matrix[76][43] = new Int(46);
    matrix[91][75] = new Complex(-1, 2);
    matrix[91][99] = new Char('b');
    cout << matrix;

    cout << "Cell (1,52) has content " << *(matrix[1][52]);

    return EXIT_SUCCESS;
}
```

You should get something like this as an output:



A screenshot of a Visual Studio 2010 debug console window. The title bar shows the file path: C:\Users\jarkkov\documents\visual studio 2010\Projects\lab13\Debug\lab13.exe. The console output displays a memory dump with the following text:

```
1: 1:          a 52:      Zilog
43: 1:          64
76: 43:         46
91: 75: (-1.0, 2.0) 99:      b
Cell (1,52) has content 52:      Zilog
```