Part I: Practice and Theory

The following problems are for practice only and will **not be collected**.

Review problems: All. **Practice Problems**: All.

Part II: Programming. The following problems will be **collected** and three of them graded. Each graded problem will be worth 25 points. Read instructions carefully!

(1) Based on **Problem P12.3**

- Consider the classes `List`, `Node`, and `Iterator` introduced in class (the implmentation is posted on CCLE). Modify the class `List` so that it contains integers instead of `strings`. Call the new class `ListInt`. You need to modify the `Iterator` and `Node` classes as well.
- Implement a member function

```
void ListInt::swap_nodes(Iterator it1, Iterator it2);
```

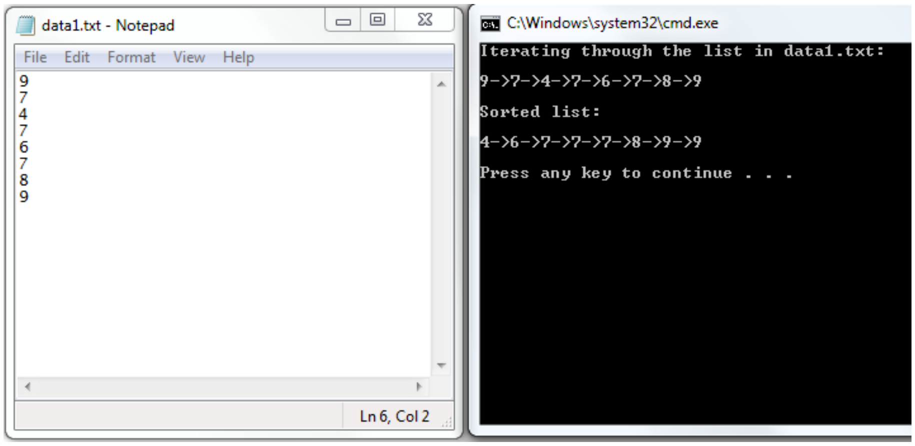
that swaps the nodes to which the iterators `it1` and `it2` point to. The implementation should NOT swap the `data`. One should switch the links instead, avoiding copying the `data`.

- Implement a member function

```
void ListInt::selection_sort();
```

that sorts the elements of the `ListInt` using selection sort algorithm (for that modify the selection sort algorithm presented on the lecture).

- Write a program that reads the list of integers from the file data1.txt into a `ListInt` and then sort the elements of the list using the member function `selection_sort()`.
- Submit the solution as hmw_6_1.cpp.
- Sample input-output:

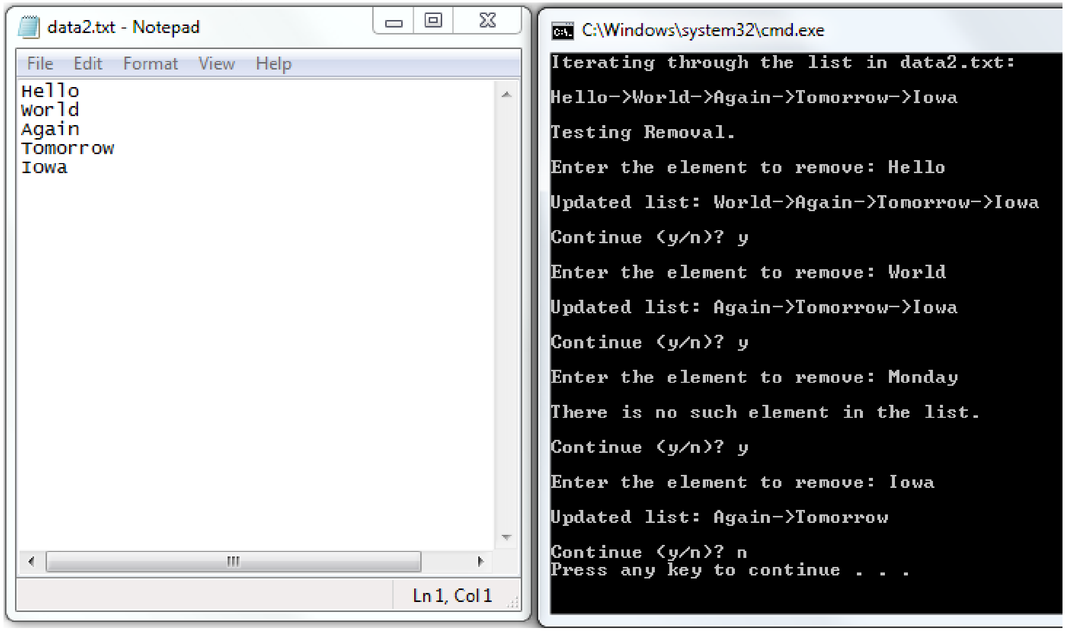


```
9
7
4
7
6
7
8
9
```

```
Iterating through the list in data1.txt:
9->7->4->7->6->7->8->9
Sorted list:
4->6->7->7->7->8->9->9
Press any key to continue . . .
```

(2) Based on **Problem P12.12**

- Turn the linked `List` of `strings` implementation into a singly-linked list `sList`: Drop the previous pointer of the nodes and the previous member function of the iterator. Reimplement the other member functions so that they have the same effect as before. Hint: In order to remove an element in constant time, iterators should store the predecessor of the current node.
- You can modify the `List` class provided in class (see the implementation posted on CCL`ccle`).
- Write a program that reads the data from `data2.txt` into the single-linked list `sList`. Display the elements of the list and then prompt the user to enter an element for removal.
- Implement a loop in which the removal action is performed until the user requests to quit.
- Submit the solution as `hmw_6_2.cpp`.
- Sample input-output:



```
Hello
World
Again
Tomorrow
Iowa
```

```
Iterating through the list in data2.txt:
Hello->World->Again->Tomorrow->Iowa
Testing Removal.
Enter the element to remove: Hello
Updated list: World->Again->Tomorrow->Iowa
Continue (y/n)? y
Enter the element to remove: World
Updated list: Again->Tomorrow->Iowa
Continue (y/n)? y
Enter the element to remove: Monday
There is no such element in the list.
Continue (y/n)? y
Enter the element to remove: Iowa
Updated list: Again->Tomorrow
Continue (y/n)? n
Press any key to continue . . .
```

(3) A `string` S consisting of N characters is considered to be properly nested if any of the following conditions is true:

- – S is empty;
- – S has the form "(U)" or "[U]" or "U" where U is a properly nested string;
- – S has the form "VW" where V and W are properly nested strings.

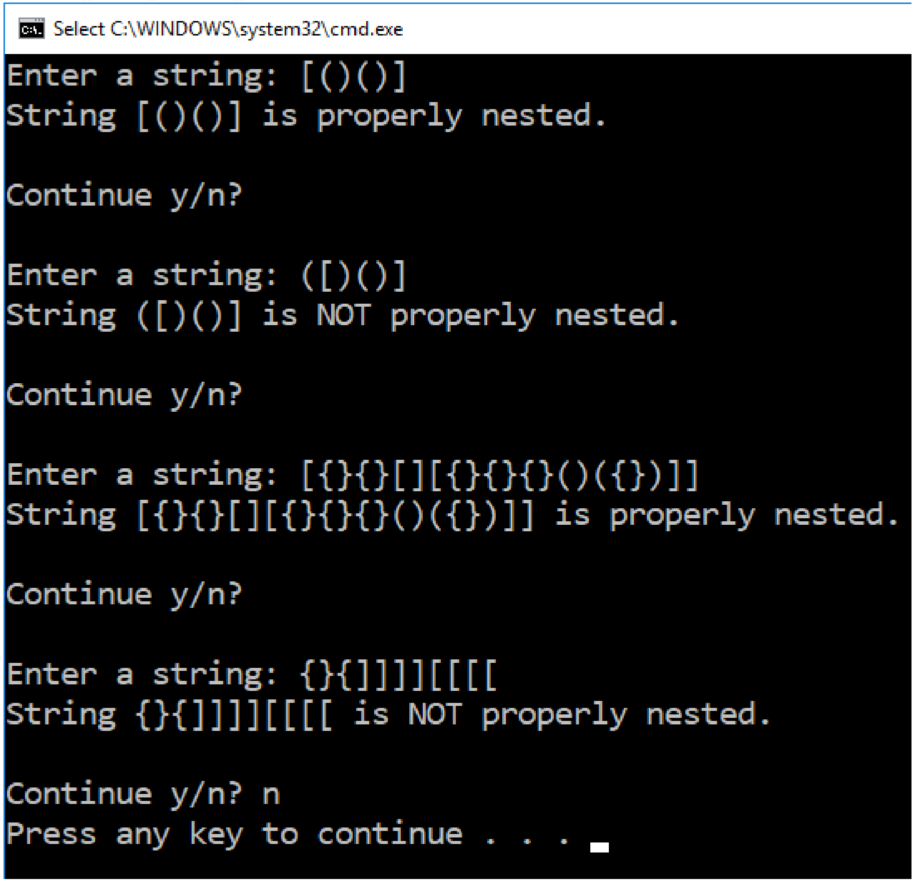
For example, the string "[()()]" is properly nested but "([)()]" is not.

- Write a function:

```
int is_nested(string S);
```

that, given a string S, returns 1 if S is properly nested and 0 otherwise. For example, given `string S = "[()()]"`, the function should return 1 and given `S = "([)()]"`, the function should return 0, as explained above.

- Write an efficient algorithm assuming that the string S consists only of the following characters: “(”, “{”, “[”, “]”, “}” and/or “)”.
- Write a program that requests a user to enter a string containing characters “(”, “{”, “[”, “]”, “}” and/or “)” and then determines whether the string is nested.
- Implement a loop in the above actions are performed until the user requests to quit.
- **Hint:** Use `stack` to solve the problem.
- Submit the solution as `hmw_6_3.cpp`.
- Sample input-output:



```
Enter a string: [()()]
String [()()] is properly nested.

Continue y/n?

Enter a string: ([)()]
String ([)()] is NOT properly nested.

Continue y/n?

Enter a string: [{}{}[][{}{}{}()({})]]
String [{}{}[][{}{}{}()({})]] is properly nested.

Continue y/n?

Enter a string: {}{]]]][[[[
String {}{]]]][[[[ is NOT properly nested.

Continue y/n? n
Press any key to continue . . .
```

(4) Based on **Problem P13.10**

- Write a program that reads a collection of `strings` (a string per line) from the file `data4.txt` and inserts them into a binary search tree. For this problem use the implementation of the class `BinarySearchTree` introduced in class (it is posted on CCLE).
- Implement a traversal member function of `BinarySearchTree` class

```
void BinarySearchTree::inorder(Action & a);
```

for inorder traversal of a binary search tree that carries out an action other than just printing the node data. The action should be supplied as a derived class of the class

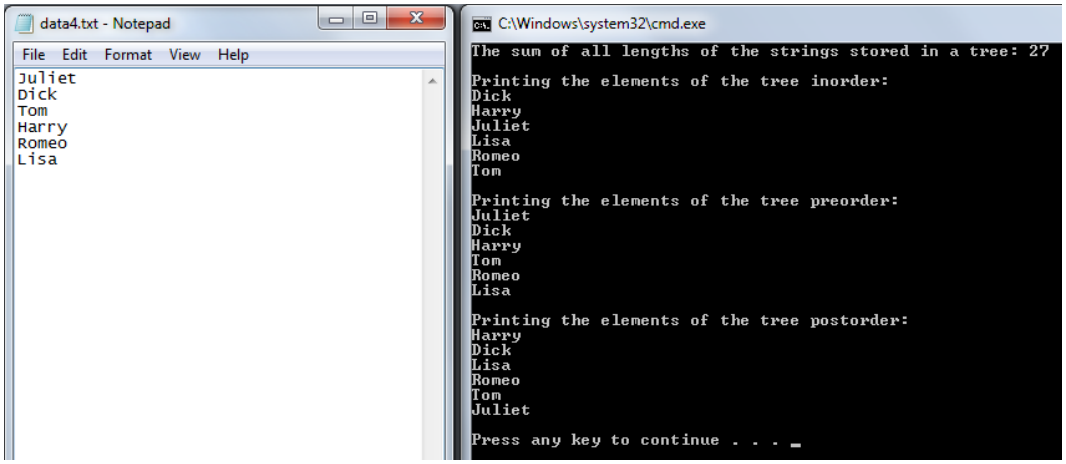
```
class Action{
public:
  void act(string str) {}
};
```

- Use the `inorder` function, and a suitable class derived from `Action`, to compute the sum of all lengths of the strings stored in a tree and then display it.
- Similarly, implement member functions

`preorder(Action & a)` and `postorder(Action & a)`

for preorder and postorder traversal of a binary search tree, respectively.

- Use the `inorder`, `preorder` and `postorder` functions and a suitable class derived from `Action`, to print the content of each `string` stored in a tree (see the sample of input-output).
- Submit the solution as `hmw_6_4.cpp`.
- Sample input-output:

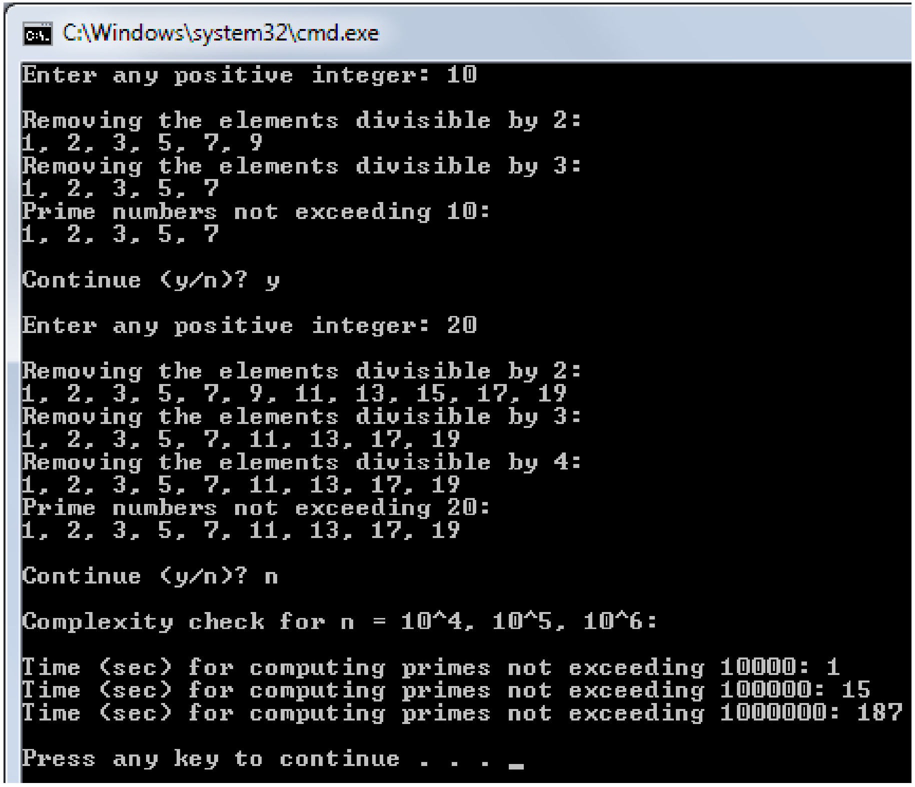


```
Juliet
Dick
Tom
Harry
Romeo
Lisa
```

```
The sum of all lengths of the strings stored in a tree: 27

Printing the elements of the tree inorder:
Dick
Harry
Juliet
Lisa
Romeo
Tom

Printing the elements of the tree preorder:
Juliet
Dick
Harry
Tom
Romeo
Lisa

Printing the elements of the tree postorder:
Harry
Dick
Lisa
Romeo
Tom
Juliet

Press any key to continue . . . _
```

(5) Based on **Problem P13.3**

- Write a program that prompts the user a positive integer $n \geq 1$ and then depicts all prime numbers not exceeding n. Use the algorithm described in Problem P13.3. At each step, after *sieving* the numbers divisible by an integer m where $1 < m < \sqrt{n}$, display the modified set (see the sample of input-output).
- Implement a loop in which the above actions is performed until the user requests to quit.
- After the loop ends check the complexity of prime number computation algorithm. Compute prime numbers not exceeding n with $n \in \{10^4, 10^5, 10^6\}$ and record the times of computations. Make sure the complexity of your algorithm does not exceed $O(n \log(n))$.
- Remark: One in fact can achieve the complexity $O(n \log(\log(n)))$, but for this one cannot use `sets` as looking up a number costs one $\log(n)$.
- Submit the solution as `hmw_6_5.cpp`.
- Sample input-output:



```
Enter any positive integer: 10

Removing the elements divisible by 2:
1, 2, 3, 5, 7, 9
Removing the elements divisible by 3:
1, 2, 3, 5, 7
Prime numbers not exceeding 10:
1, 2, 3, 5, 7

Continue (y/n)? y

Enter any positive integer: 20

Removing the elements divisible by 2:
1, 2, 3, 5, 7, 9, 11, 13, 15, 17, 19
Removing the elements divisible by 3:
1, 2, 3, 5, 7, 11, 13, 17, 19
Removing the elements divisible by 4:
1, 2, 3, 5, 7, 11, 13, 17, 19
Prime numbers not exceeding 20:
1, 2, 3, 5, 7, 11, 13, 17, 19

Continue (y/n)? n

Complexity check for n = 10^4, 10^5, 10^6:

Time (sec) for computing primes not exceeding 10000: 1
Time (sec) for computing primes not exceeding 100000: 15
Time (sec) for computing primes not exceeding 1000000: 187

Press any key to continue . . .
```