Problem 1

Given two ordered lists A and B, write a function *intersection* that returns a new ordered list with their common elements. None of the two lists have repeated elements.

For example, given the lists $2, 3, 6, 9$ and $2, 5, 9$, the result should be $2, 9$.

```
#include <list>

list<int> intersection(const list<int>& A, const list<int>& B) {
    // WRITE YOUR CODE HERE
}
```

Problem 2

Given an ordered list of integers, write a function *largest_null_segment* that returns the size of its largest null segment. A null segment of the list is a compact sublist in which the sum of its elements is 0.

For example, given the list: $-9, -7, -6, -4, -3, -1, 3, 5, 6, 8, 9$, its largest null segment contains seven elements (i.e., $-6, -4, -3, -1, 3, 5, 6$). Note that the segment $-4, -3, -1, 3, 5$ is also null, but it's not the largest one.

```
#include <list>

int largest_null_segment(const list<int>& A) {
    // WRITE YOUR CODE HERE
}
```

Problem 3

Given a list of n integers, we want to play the following game:

- initialy, all integers are 1;
- each time the user introduces a pair of values (x, y) where x is an integer and y is a boolean, you have to find the first integer i in the list such that the sum from the first element in the list to that element is bigger than x. Then, if y is $true$, you have to increase that integer by 1, and if y is $false$, you have to decrease that integer by 1.
- as we want to avoid elements in the list equal to zero, when this happens you have to add one to each integer.

Problem 3 continued on next page. . .

Complete the code given below taking into account that x is in the range $[0, s)$ (s being the current sum of integers in the list), and assuming that we will always have at least one element in the list.

```
#include <list>
#include <iostream>
using namespace std;

// Pre: n >= 0
// Post: returns a list of n 1's.
list<int> create_list_with_1(int n) {
     // WRITE YOUR CODE HERE
}

// Pre: lst is a valid list,
//      x in the range [0, s), where s is the sum of elements in the list
//      y is 0/1
// Post: lst is modified as described in the statement
void modify_list(list<int>& lst, int x, bool y) {
     // WRITE YOUR CODE HERE
}

void print(const list<int>& l) {
    list<int>::const_iterator i = l.begin();
    while (i != l.end()) {
        cout << *i << endl;
        ++i;
    }
}

// The input is the number of elements of the list (at least one),
// and then as many pairs of integers x y as you want
int main() {
     int n;
     cin >> n;
     list<int> lst = create_list_with_1(n);
     int x;
     bool y;
     while (cin >> x >> y) {
         modify_list(lst, x, y);
     }
     print(lst);
}
```

Problem 4

Write a function that checks the correct parenthesization of a given word by using a stack of characters.

```
#include <stack>

bool correct(const string& w) {
    // WRITE YOUR CODE HERE
    // you can access each char as w[position]
    // the size of w is w.size()
}

int main() {
    string word;
    while (cin >> word) {
        if (correct(word)) cout << word << " is correct." << endl;
        else cout << word << " is not correct." << endl;
    }
}
```

Problem 5

Consider this program (whose inclusions have been removed):

```
void print(int n) {
    if (n > 0) {
        cout << ' ' << n;
        print(n - 1);
        print(n - 1);
    }
}

int main() {
    int n;
    while (cin >> n) {
        print(n);
        cout << endl;
    }
}
```

Take a look at the sample input and sample output to see what this program prints for every given number.

Problem 5 continued on next page...

Without modifying the main(), reimplement the procedure print(n) with no calls at all, recursive or not, so that the output of the program does not change.

The input consists of several strictly positive natural numbers. For every number, print a line identical to the one written by the program above.

Observation: to solve this exercise, the only containers that you should use are stacks.

Input	Output
1	1
2	2 1 1
3	3 2 1 1 2 1 1
4	4 3 2 1 1 2 1 1 3 2 1 1 2 1 1