Lab 5.3.5 Singly linked list: part 5

Objectives

Familiarize the student with:

- implementing data structures in C++;
- traversing data structures;
- keeping data structures consistent at all times.

Scenario

To make the interface somewhat symmetrical, we should add a method for removing the last element of the list.

Unfortunately, there's no simple trick to avoid traversing the list.

The tail pointer tells us which element we'll be removing, but to remove it we must modify the previous element, which we need to find, starting at the head.

Add the implementation of the pop_back method to List.

```
#include <iostream>

using namespace std;

class Node
{
public:
  Node(int val);
  int value;
  Node* next;
};

class List
{
public:
  List();
  void push_front(int value);
  bool pop_front(int "value);
  void push_back(int value);
  bool pop_back(int "value);
  int size()
private:
  // other members you may have used
  Node* head;
  Node* tail;
};

List::List() : head(nullptr), tail(nullptr)
{
}

bool List::pop_back(int "value)
{
  // implement me!
  return false
}

// ...

int main()
{
  List list;
  //
  list.push_front(1);
  list.push_front(2);
  list.push_front(3);
  list.push_front(4);

  int value = 0;
  while (list.pop_back(value))
  {
    cout << value << endl;
  }
  return 0;
}
```

Example input

Example output

```
1
2
3
4
```