

Lab 5.3.4 Singly linked list: part 4

Objectives

Familiarize the student with:

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

Scenario

Now, our list implementation can be easily used to create a stack implementation, since we can push and pop items to the front of the list.

Another common data structure is the queue. A queue allows us to add elements to one side of the data structure, but remove them only from the other side.

So, queues allow us to process items in order of appearance, which is often considered fair. This is a common scenario in many cases:

- when handling calls from customers, we want to answer the calls in order, so we don't make one of our customers wait forever;
- ticket reservation systems should process requests in order of appearance, so someone who wanted the ticket first should get it.

To enable us to use our list as a queue, we'll add a new method to it: `push_back()`.

One strategy to add an element to the back of the list would be to always search for the last element and add the new element to the list.

This, unfortunately, is a lot of work just to add a single element.

Another approach is to create a new pointer that will always point to the last node in the list. Let's try this approach.

```
// An empty list:
//
// Node*
// +-----+
// | head |-->nullptr
// +-----+
//
// Node*
// +-----+
// | tail |-->nullptr
// +-----+
//
//
//
// A list with two elements:
//
// Node*      Node      Node
// +-----+  +-----+  +-----+
// | head |-->|value|  +-->|value|
// +-----+  +-----+  +-----+
//           |next |--+  |next |-->nullptr
//           +-----+  +-----+
//                   ^
//                   |
// +-----+
// | tail |-----+
// +-----+
#include <iostream>

using namespace std;

class Node
{
    ...
}
```

```

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

Node::Node(int val) : value(val), next(nullptr)
{
}

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

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

void List::push_front(int value)
{
    // You need to fix this part!
    // The tail pointer needs to be modified only when the first element is added
    Node* new_head = new Node(value);
    new_head->next = head;
    head=new_head;
}

// All of your previously written methods may require a little fixing

// START
// +-----+ +-----+ +-----+
// | head |-->| X | +-->| Y |
// +-----+ +-----+ | +-----+
//          |next |--+ |next |-->nullptr
//          +-----+ +-----+
//                      ^
//          +-----+ |
// | tail |-----+
// +-----+
//
// STEP 1
//
//          new Node
// +-----+ +-----+ +-----+ +-----+
// | head |-->| X | +-->| Y |          | Z |
// +-----+ +-----+ | +-----+ +-----+
//          |next |--+ |next |-->nullptr |next |-->nullptr
//          +-----+ +-----+ +-----+
//                      ^
//          +-----+ |
// | tail |-----+
// +-----+
//
// STEP 2
//

```

```

//
// +-----+ +-----+ +-----+ +-----+
// | head |-->| X | +-->| Y | +-->| Z |
// +-----+ +-----+ | +-----+ | +-----+
//          |next |--+ |next |--+ |next |-->nullptr
//          +-----+ +-----+ +-----+
//
//                      ^
// +-----+          |
// | tail |-----+
// +-----+
//
// STEP 3
// +-----+ +-----+ +-----+ +-----+
// | head |-->| X | +-->| Y | +-->| Z |
// +-----+ +-----+ | +-----+ | +-----+
//          |next |--+ |next |--+ |next |-->nullptr
//          +-----+ +-----+ +-----+
//
//                      ^
// +-----+          |
// | tail |-----+
// +-----+
void List::push_back(int value)
{
    // implement me!
}

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

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

```

Example input

Example output

```

1
2
3
4

```