

Lab 5.3.1 Singly linked list: part 1

Objectives

Familiarize the student with:

- implementing data structures in C++;
- dynamic allocation of C++ objects.

Scenario

There are some classic data structures in computer science. One example of this that you've seen in the lectures is called the stack.

In the next few exercises, we'll build yet another classic data structure, the linked list.

To be exact, we'll be building the singly linked list.

The building block of a singly linked list is a node, which consists of two parts:

- a value;
- a link to the next node in the list, also known as the "next" pointer.

The beginning of a list is called the head and it's a pointer to the first node in the list.

When the list is empty, the head usually points to nothing, i.e. the nullptr.

Similarly, if a node is the last node in the list, the "next" pointer of that node will point to nullptr.

Let's build a list of integers.

Our initial implementation of the list should have two methods:

- push_front, which will add a value to the front of the list;
- pop_front, which will return and remove the value.

```
// An empty list:
//
// Node*
// +-----+
// | head |-->nullptr
// +-----+
//
//
//
// A list with two elements:
//
// Node*      Node      Node
// +-----+ +-----+ +-----+
// | head |-->|value| +-->|value|
// +-----+ +-----+ | +-----+
//           |next |--+ |next |-->nullptr
//           +-----+ +-----+
//
//
#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");
private:
    Node* head;
};

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

void List::push_front(int value)
{
    Node* new_head = new Node(value);
    new_head->next = head;
    head=new_head;
}

// START
// +-----+ +-----+ +-----+
// | head |-->| X | +-->| Y |
// +-----+ +-----+ | +-----+
//          |next |--+ |next |-->nullptr
//          +-----+ +-----+
//
// STEP 1
//
//          +-----+
//          |popped|
//          +-----+
//          |
//          V
// +-----+ +-----+ +-----+
// | head |-->| X | +-->| Y |
// +-----+ +-----+ | +-----+
//          |next |--+ |next |-->nullptr
//          +-----+ +-----+
//
// STEP 2
// +-----+
// | head |-----+
// +-----+          |
//          V
// +-----+ +-----+ +-----+
// |popped|-->| X | +-->| Y |
// +-----+ +-----+ | +-----+
//          |next |--+ |next |-->nullptr
//          +-----+ +-----+
//
// STEP 3
// returned = popped->value;
// delete popped;
// +-----+ +-----+
// | head |-->| Y |

```

```
// +-----+ +-----+
//          |next |-->nullptr
//          +-----+

bool List::pop_front(int "value")
{
    // implement the pop
    // don't forget to delete the popped node!
    // and fix the return value
    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_front(value))
    {
        cout << value << endl;
    }
    return 0;
}
```

Example input

Example output

```
4
3
2
1
```