

Lab 5.3.8 Doubly linked list

Objectives

Familiarize the student with:

- implementing data structures in C++;
- dynamic allocation of C++ objects;
- preventing memory leaks and deallocating acquired resources;
- providing derived data about the implemented data structure;
- keeping the data structure consistent at all times;
- traversing data structures;
- accessing data stored in data structures;
- creating copies of data structures;
- implementing and using copy constructors.

Scenario

Now that we've finished working on our singly linked list, you can practice on your own, building a completely new type of list!

This time, you'll build a doubly-linked list. The only difference is that all nodes now have a reference to the previous node as well.

This will make moving back through the list a lot easier, not to mention removing the last element.

Your new list should have the same methods as the previous one, so this task may take you some time.

```

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

```

```

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

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

```