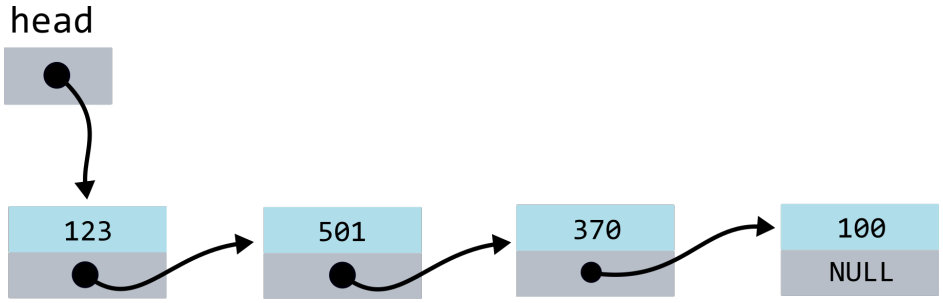
Семинар #11: Связный список.

Создадим структуру `Node`, которая будет содержать:

- **Данные**: одно или несколько полей каких-угодно типов. В данном случае это 1 переменная `value`.
- **Связь**: указатель `next` на структуру того же типа `Node`.


```c
struct node
{
    int value;
    struct node* next;
};
typedef struct node Node;
```

Используя такую структуру, можно создать Связный список:



Примечание: `NULL` - это просто константа равная 0. Её используют вместо нуля для указателей, чтобы различать числовые переменные и указатели.

Вычислительные сложности операций со списком:

Операция	Массив	Односвязный список
Доступ по номеру	$O(1)$	$O(N)$
Поиск	$O(N)$	$O(N)$
Вставка в начало	$O(N)$	$O(1)$
Вставка в конец	$O(1)$	$O(N)$
Вставка в конец если известен указатель на последний элемент	$O(1)$	$O(1)$
Вставка в середину	$O(N)$	$O(N)$
Вставка в середину если известен указатель на предыдущий элемент	$O(N)$	$O(1)$

Чему равны вычислительные сложности следующих операций:

- Нахождение размера списка. Что можно сделать, чтобы нахождение размера выполнялось быстрее?
- Удаление элемента из начала списка.
- Удаление элемента из конца списка. Что нужно знать, чтобы эта операция выполнялась быстрее?
- Удаление элемента из середины списка. Что нужно знать, чтобы эта операция выполнялась быстрее?

Задачи. Напишите следующие функции для работы со связным списком:

Начальный код в файле `list.c`.

1. `Node* list_create()` – инициализирует список (создаёт и возвращает список нулевого размера). Эта функция просто возвращает `NULL`. Зачем нужна такая простая функция? Она нужна для согласованности с реализациями других структур данных. Например, при реализации хеш-таблицы у нас будет более сложная функция `hashtable_create`, которая будет создавать хеш-таблицу.
2. `void list_add_first(Node** p_head, int x)` – добавляет элемент x в начало списка. Чтобы добавить элемент, нужно для начала выделить необходимое количество памяти под этот элемент, затем задать поля нового элемента таким образом, чтобы он указывал на начало списка. В конце нужно поменять значение указателя на начало списка. Обратите внимание, что так как нужно изменить значение указателя, то в эту функцию нужно передавать указатель на указатель.
3. `void list_add_last(Node** p_head, int x)` – добавляет элемент x в конец списка. (решение этой задачи есть в файле `list.c`).
4. `int list_remove_first(Node** p_head)` – удаляет элемент из начала списка и возвращает его значение. Не забудьте изменить `*p_head`.
5. `int list_remove_last(Node** p_head)` – удаляет элемент из конца списка и возвращает его значение.
6. `void list_print(const Node* head)` – распечатывает все элементы списка.
7. `int list_size(const Node* head)` – возвращает количество элементов списка.
8. `int list_destroy(Node* head)` – освобождает всю память, выделенную под список. Так как память выделялась под каждый элемент отдельно, то освобождать нужно также каждый элемент по отдельности.
9. `void list_reverse(Node** p_head)` – переворачивает связный список. Первый элемент становится последним, а последний первым. В данной задаче вам не нужно перемещать элементы `value` или сами структуры. Нужно просто изменить указатели.
10. `void list_concatenate(Node** p_head1, Node** p_head2)`, которая добавляет второй связный список в конец первого.
11. Реализовать абстрактный тип данных стек(Stack) на основе связного списка.
12. Создайте связный список размера 10. Последний элемент должен ссылаться на 5-й. Что будет, если попробовать напечатать такой список?
13. Написать функцию `int list_is_loop(Node* head)`, которая проверяет, если в связном списке цикл.
14. Написать функцию `int list_fix_loop(Node* head)`, которая проверяет, если в связном списке цикл. И если цикл есть, то она размыкает его.