Абстрактные типы данных

АТД

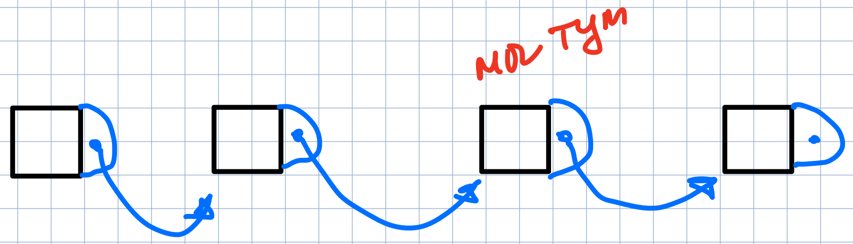
мы сделали сами

Я. прогр.

типы данных

(I) АТД список. (односвязный список)

<u>Порисуем</u>



"Мы тут" $\longrightarrow$ перейди дальше

✓ 1. PushBack

✓ 2. Push Front

✓ 3. PopBack

✓ 4. Pop front

✓ 5. Size

6. Add (index, elem)

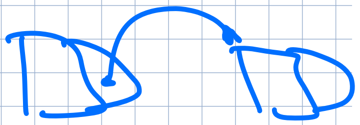
7. Remove (index)

8. delete ()

9. (6-7) -сделать правильно

Напоминание:

```
struct Node {
    int value;
    Node* next;
};
```

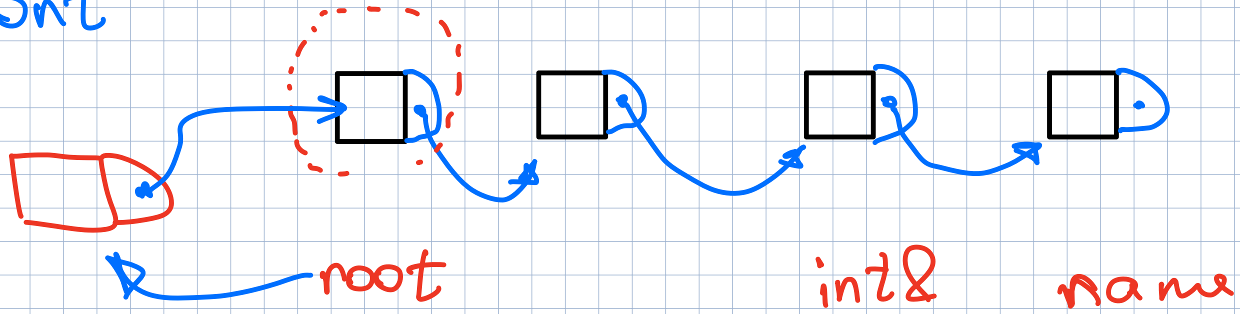


```
Node new_name;
new_name.value = 10;
new_name.next = new Node;
```

$* [\underbrace{\text{new\_name.next}}_{\text{Node*}}]\text{.value} = 7;$

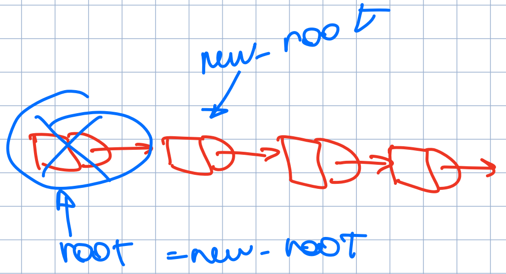
new_name.next → value = 7

Ⅰ Push Front

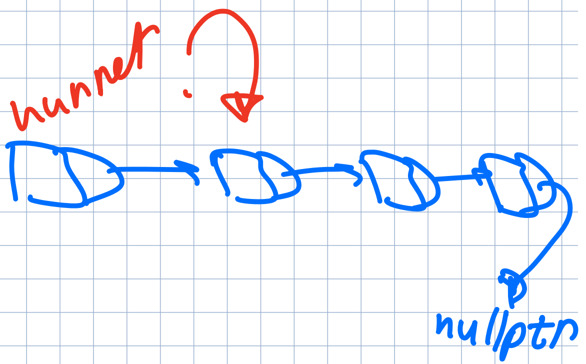


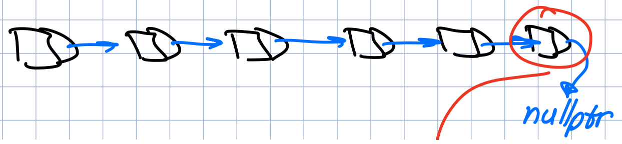
```
void  PushFront( int new value , Node*& root)
{
    Node* new_node = new Node;
    new_node -> value = new_value;
    new_node -> next = root;
    root = new_node;
}
```

Ⅱ Pop Front (Node*& root) {



```
    if   root == nullptr
                  ↳ return

    Node*  new_root = root → nexT;
    delete  root;
    root =  new_root;

}

(II)  Size (const  Node*&  root ) {
        count = 0
       Node*   runner = root;
       while   runner != nullptr
       {
            count ++;
            runner  =  runner → next;
       }

       return   count
}
```



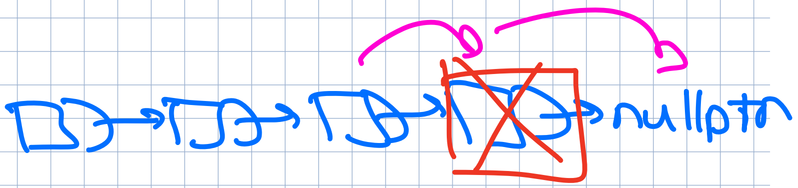


IV

[]->nullptr

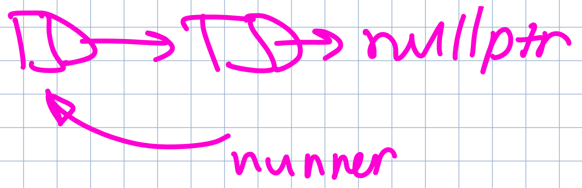
```
PushBack ( int new_value , Node*& root ) {
проверка пустоты
    if root == nullptr
        ↳ PushFront( new_value , root )
          return
дошли до конца
    Node* runner = root;
    while runner -> next != nullptr
          ↳ runner = runner -> next
новый эл-т
    Node* new_node = new Node;
    new_node -> value = new_value;
    new_node -> next = nullptr;
    runner -> next = new_node;
}
```

runner=nullptr



(V) Pop Back (Node*& root) {

дедушка папа



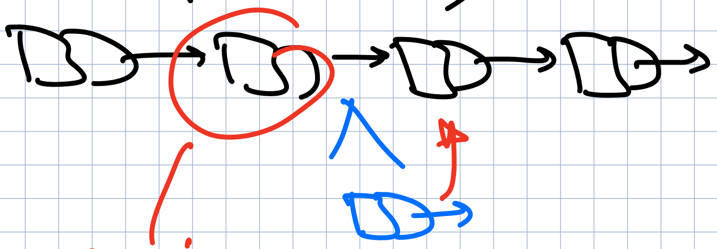
крайние случаи

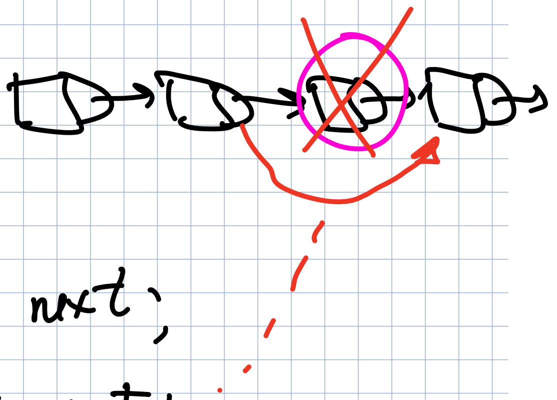
```
if   root == nullptr
        ↳ return

if   root → next == nullprt
        Popfront (root)
```

Бежим до конца

```
Node*  runner = root;
while  runner → next → next != nullptr:
            runner = runner → next;

delete  runner → next;        // "удалили папу"
runner → next = nullptr;      // "забыли папу"

}
```

```
(vi)   Add  ( int newvalue , Node* & previous )
{
    Node* new_node = new Node;
    new_node -> value = new_value;
    new_node -> next = previous -> next;
    previous -> next = new_node;
}
```

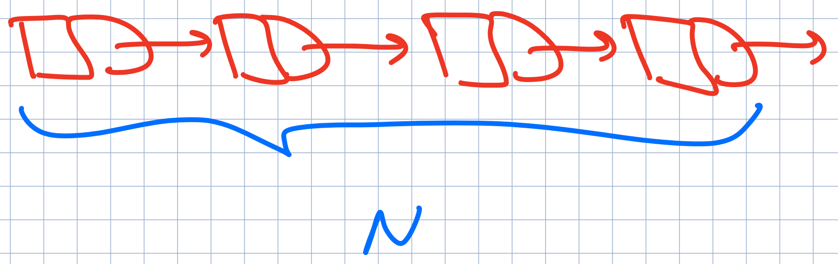




```
(vii)   Remove ( Node* & previous )
{
        Node* current = previous -> next;
        previous -> next = current -> next;
        delete current;
}
```

VIII

```
Delete ( Node*& root)
{
    Node* runner = root ;
    while  runner != nullptr {
           Node* next = runner→next;
           delete  runner;
           rener = next
    }
    root= nullptr;
}
```



1. PushBack $O(N)$
2. Push Front $O(1)$
3. PopBack $O(N)$
4. Pop front $O(1)$
5. Size $O(N)$

$O(1)$ 6. Add (index, elem)
$O(1)$ 7. Remove (index)
$O(N)$ 8. delete ()
9. (6-7) - сделать правильно

плохо: тея обращения по индексам

```
Node* end        // указатель на последний.
```

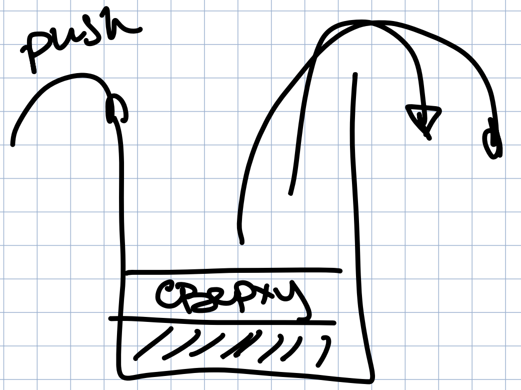
Дек - двусвязный список

```
main () {
    Node* root = new Node;
    PushBack (10, root)
    . - . - .
    Delete (root)
}
```

АТД Стек

LIFO

Last In First Out



на списък Node* root ...

```
push ( Node*& root, int elem )
        PushFront( elem, root )
```
O(1)

```
pop ( Node*& root )
        PopFront ( root )
```
O(1)

```
top ( Node*& root )
    if  root == nullptr:
        ↳ return  -1;
```
O(1)

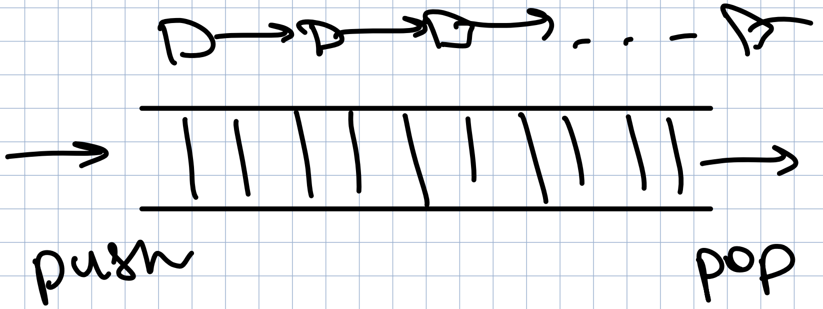
renurn root → value

$O(1)$ is_empty (Node*& root)
↳ return root == nullptr

АТД Очередь

FIFO

first in
first out



push-queue (Node*& root, new_value

$O(1)$ Push front (root, new_value)

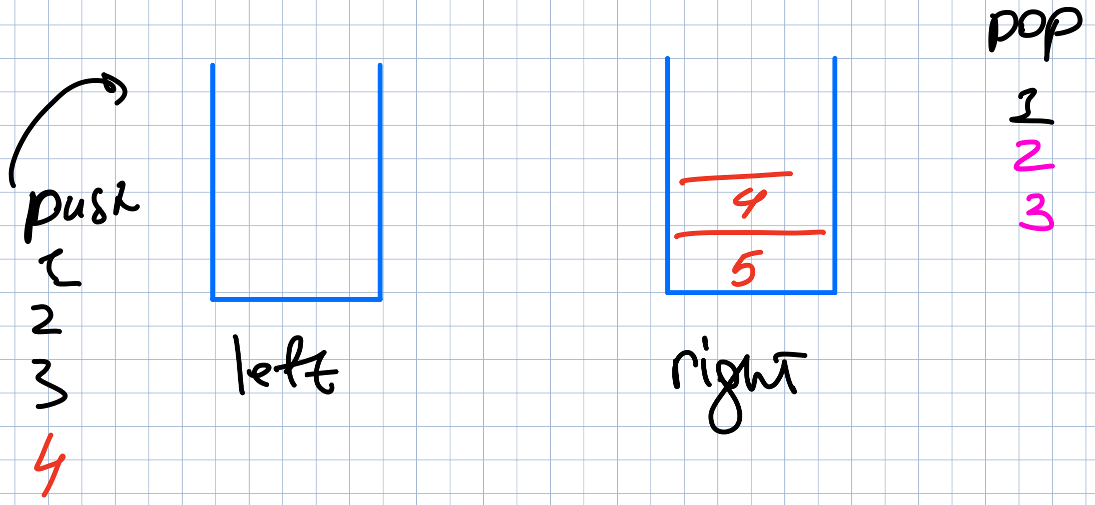
pop-queue (Node*& root)

~~$O(n)$~~ $O(1)$ Pop Back (root)

(уииии :)

АТД Очередь (на $2^х$ стеках)

Идея:



1) если right <u>не</u> пуст $\Rightarrow$ pop берет из right $O(1)$

2) если righ <u>пуст</u> $\rightarrow$ перекладываем <u>все</u> из left $\rightarrow$ right $O(N)$
pop из right

3) push всегда в ~~left~~ $\underline{O}(1)$

происходит **редко!**

раз в N "обращений" pop

$$\frac{1}{N}\,\underline{O}(n) = \underbrace{\underline{O}(1)}_{\text{амортизированно}}$$

амортизированно $\underline{O}(1)$