LINKED LIST (Связный список)
Односвязные

Положение элемента определяется указателем на следущий или предыдущий элемент.
Связные списки могут различаться. Есть односвязные списки, в которых каждый узел хранит указатель только на следующий узел. Есть двусвязные списки: в них каждый элемент хранит ссылку как на следующий элемент, так и на предыдущий. Есть кольцевые замкнутые списки. В данном случае мы рассмотрим создание односвязного списка.

1. **Определяем единичный узел.**

```
public class Node<T>
{
    public Node(T data)
    {
        Data = data;
    }
    public T Data { get; set; }
    public Node<T> Next { get; set; }
}
```

Класс Node - обобщенный, хранит данные любого типа(дженерик). Для хранения данных есть **свойство Data.** Для перехода на следующий узел определено свойство **Next.**

```
Node<T> head; // головной/первый элемент
Node<T> tail; // последний/хвостовой элемент
int count;  // количество элементов в списке
```

```
Важно определить 3 переменные:
1 - головной(первый, head) элемент;
2 - последний(хвостовой, tail) элемент;
3 - количество элементов
```

Кольцевой (круговые, циклические)- списки являются разновидностью связных списков. Они могут быть односвязными или двусвязными. Их отличительной особенностью является то, что условной последний элемент хранит ссылку на первый элемент, поэтому список получается замкнутым или кольцевым. Здесь нам также потребуются ссылки на условно первый и

последний элемент в виде переменных head и tail, хотя формально список будет кольцевым не будет иметь начала и конца.

Кольцевой двусвязный список - представляет замкнутый список, в котором указатель на элемент может перемещаться как вперед, так и назад по кругу.
Каждый узел такого списка опять же будет представлять элемент, который хранит указатели на следующий и предыдущий узлы:
Однако несмотря на то, что формально список замкнут, и у него нет начала и конца, все равно для некоторого базового отчета в таком списке будет храниться ссылка на первый элемент, относительно которого будет идти добавление новых элементов. В тоже время в отличие от кольцевого односвязного списка теперь уже не надо хранить указатель на формально последний элемент списка.

DoublyLinkedList (Двусвязный список)

Двусвязные списки также представляют последовательность связанных узлов, однако теперь каждый узел хранит ссылку на следующий и на предыдущий элементы.

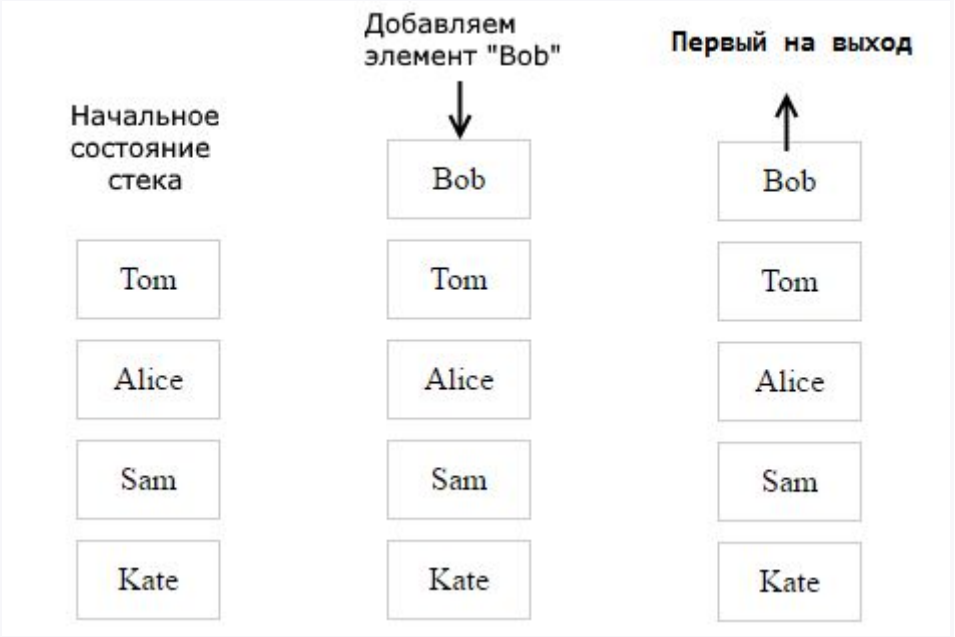
```
public class DoublyNode<T>
{
    public DoublyNode(T data)
    {
        Data = data;
    }
    public T Data { get; set; }
    public DoublyNode<T> Previous { get; set; }
    public DoublyNode<T> Next { get; set; }
}
```

```
Важно определить 3 переменные:
1 - Узел
2 - Ссылка на предыдущий элемент
3 - Ссылка на следующий элемент
```

Двунаправленность списка приходится учитывать при добавлении или удалении элемента, так как кроме ссылки на следующий элемент надо устанавливать и ссылку на предыдущий. Но в то же время у нас появляется возможность обходить список как от первого к последнему элементу, так и наоборот - от последнего к первому элементу. В остальном двусвязный список ничем не будет отличаться от односвязного списка.

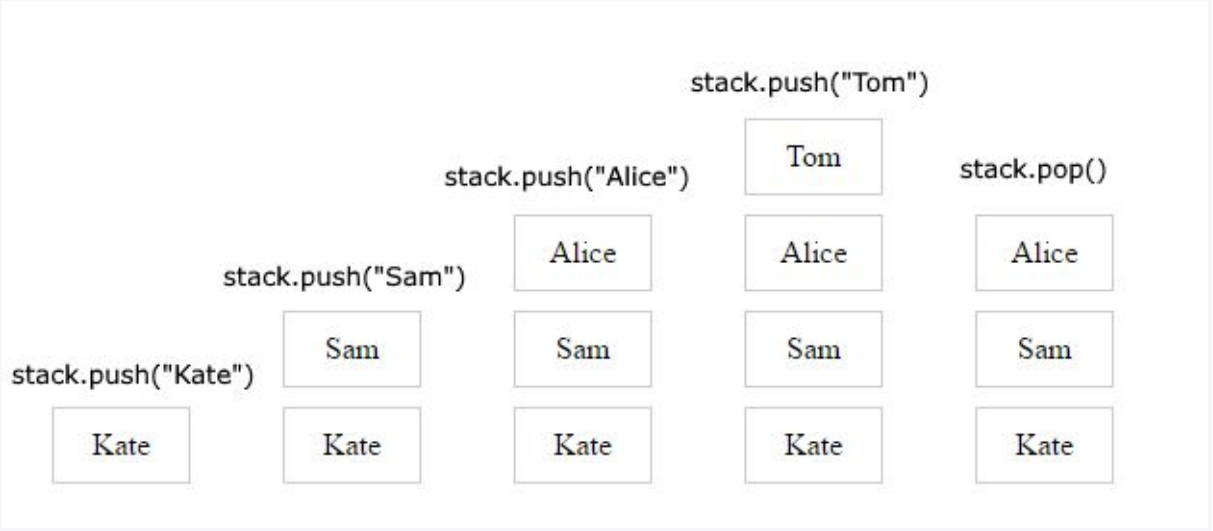
Стек на основе массива (System.Collections.Generic.Stack)

Стек представляет собой структуру данных, которая работает по принципу LIFO (Last In First Out - "последний пришел - первый вышел").



Вершина стека - последний элемент. Структура стек обладает стандартным функционалом, который составляют метод добавления элемента (как правило, называется push()) и метод извлечения элемента из вершины

стека (обычно называется pop()). Кроме того, нередко реализации стеков содержат метод получения элемента из вершины без его извлечения, метод определения размера стека и ряд других.



Очередь(Queue)

Очередь (queue) - это структура данных, которая работает по принципу FIFO (First In First Out - Первый пришел, первый вышел). При добавлении новый элемент помещается в конец очереди или ее хвост, а удаление идет с начала очереди или головы очереди.

```
public class Node<T>
{
    public Node(T data)
    {
        Data = data;
    }
    public T Data { get; set; } - данные
    public Node<T> Next { get; set; } - ссылка на следующий элемент
}
```

Для добавления в очередь нам надо переустановить ссылку на последний элемент tail.
При удалении надо переустановить ссылку на первый элемент. Так как первый элемент удаляется, то новым первым элементом становится следующий за ним.

Дек(DeQueue)

Дек (deque) представляет двустороннюю очередь, в которой элементы можно добавлять как в начало, так и в конец. Удаление также может идти как с начала, так и с конца.

Поскольку реализовать добавление и удаление нужно с двух сторон, то за основу реализации можно взять организацию двусвязного список.

В отличие от класса очереди здесь определены два метода для добавления и два для удаления. Ссылка указывает либо на head либо на tail.

На C# реализуется через сторонние библиотеки.