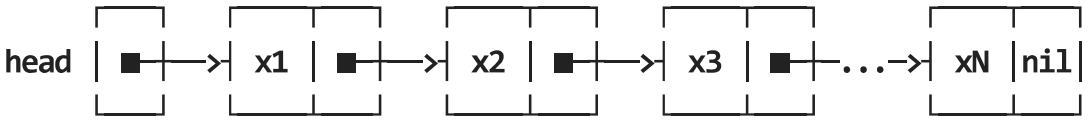
LIST

```
type list struct {
    head *lmnt
}
```

или

```
type list struct {
    head *lmnt
    len int
}
```

```
type lmnt struct {
    x data
    next *lmnt
}
```



Элементы списка - типа lmnt; x1, x2 ... - данные. Их тип - тип data - описывается при необходимости в каждом конкретном случае. Список может быть и пустым. В этом случае head имеет значение nil.

Сразу одно замечание:
ниже имеется довольно много методов, связанных с перестановкой элементов списка, с изменением порядка их следования. Действовать во всех таких ситуациях надо естественно, а именно
переставлять элементы, изменяя ссылки, а не переприсваивая значения данных внутри элементов; допускается только изменение значений указателей внутри элементов - полей next

01.

```
func (l list) Max () int
type data int
```

Максимум списка.
02.

```
func (l list) Min () int
type data int
```

Минимум списка.
03.

```
func (l list) CompareList ( l2 list) bool
type data int
```

Совпадает ли список l2 с ресивером l.
04.

```
func (l *list) InsertKth (x data, k int)
```

Вставить элемент x на k-ю позицию, k > 0. Считаем, что элементы в списке пронумерованы с головы списка, и нумерация начинается с 0. Можно предусмотреть, что функция будет возвращать ошибку, если k слишком большое - больше длины списка. Либо возвращать bool - удалось или не удалось вставить.
05.

```
func (l list) GetKth (k int) data
```

Возвращает k-й элемент списка, k >= 0. Как и в предыдущей функции, считаем, что элементы в списке пронумерованы с головы списка, и нумерация начинается с 0. Можно предусмотреть, что функция будет возвращать ошибку, если k слишком большое - больше длины списка. Либо возвращать bool - удалось или не удалось вставить.
06.

```
func (l *list) RemoveKth (k int) bool
```

Удаляет из списка k-й элемент, k >= 0. И здесь, считаем, что элементы в списке пронумерованы с головы списка, и нумерация начинается с 0. Но тут уж точно функция должна возвращать, был в списке элемент с таким номером и его удалось убрать из списка, или нет.
07.

```
func (l list) CopyList list
```

Создает в куче копию списка и возвращает его.
08.

```
func (l *list) ConcatList (list2 list)
```

Прицепляет список list2 к концу списка l.
09.

```
func (l list) First (x data) int
```

Вернуть номер первого элемента списка, равного x. Если такого элемента в списке нет, то вернуть -1.
10.

```
func (l list) Last (x data) int
```

Вернуть номер последнего элемента списка, равного x. Если такого элемента в списке нет, то вернуть -1.
11.

```
func (l *list) Truncate (k int)
```

Отсекает все элементы списка, начиная с k-го. Если k<0 или больше номера последнего элемента списка, то ничего не делает.
12.

```
func (l *list) TruncateFromData (x data)
```

Отсекает все элементы списка, начиная с первого элемента со значением x.
13.

```
func (l *list) DeleteX (x data);
```

Исключить из списка все элементы, равные x.
14.

```
func  (l *list) Revers ()
```

Развернуть список задом наперёд. Держим после этого список за новый начальный - бывший последний - элемент.
15.

```
func  (l *list) RotatePlus ()
```

Переставляет первый элемент в конец списка.
16.

```
func  (l *list) RotatePlusK (k int)
```

Переставляет первые k элементов в конец списка.
17.

```
func  (l *list) RotateMinus ()
```

Переставляет последний элемент в начало списка.
18.

```
func  (l *list) RotateMinusK (k int)
```

Переставляет последние k элементов в начало списка.
19.

```
func (l *list) SeparateOnParity ()
type data = int
```

Переставляет элементы списка так, что сначала идут все четные числа (в том порядке, как они встречаются в исходном списке), затем - все нечетные (в таком же порядке).
20.

```
func (l list) Unambiguous () list
```

Создает новый список, в который входят только неповторяющиеся элементы исходного списка.
21.

```
func (l *list) Shuffle ()
```

Переставляет элементы списка в следующем порядке: сначала последний, затем первый, затем предпоследний, затем второй, за ним - третий с конца и т.д.
22.

```
func (l *list) Transposition (j, k int)
```

Переставляет j-й и k-й элементы списка, если такие в нем есть.
23.

```
func (l *list) ReverseSegment (j, k int)
```

Изменяет ссылки так, чтобы элементы от j-го до k-го шли в обратном порядке. Если хотя бы одного элемента (j-го или k-го) в списке нет или если не выполняется условие j<k, то ничего не делает.
24.

```
func (l list) CountX (x int) int
type data int
```

Количество вхождений числа x в список.
25.

```
func (l list) SubstX (x int, y int)
```

```
type data int
```

Заменяет все элементы, равные x, на y.
26.

```
func (l list) SubList (sublist list) (bool, int)
```

Если список sublist входит (подряд) в список l, то вернуть true и номер элемента списка l, с которого начинается первое такое вхождение. В противном случае возвращать false и -1.
27.

```
func (l list) CountSub (sublist list) int
```

Возвращает количество вхождений списка sublist2 в список l. Вхождения могут перекрываться.
28.

```
func (l list) SubSequence (sublist list) bool
```

Входит ли список sublist в список l как подпоследовательность?
Список A входит в список B как подпоследовательность, если все элементы списка A входят в список B, причем в том же порядке, хотя и не обязательно подряд. Например, список 1,3,5 входит в список 2,1,4,3,3,5, но не входит в список 2,1,4,5,3,3.
29.

```
func (l *list) EraseRepetitions ()
```

Удаляет все повторяющиеся элементы списка, оставляя только первое вхождение каждого элемента.
30.

```
func (l *list) PurgeRepetitions1 ()
```

Если подряд идут несколько одинаковых элементов, то убирает все повторы, оставляя от них только один элемент. Например, список 2 5 5 3 2 5 2 2 2 2

становится списком 2 5 3 2 5 2.
31.

```
func (l *list) Intersect (pattern list)
```

Оставляет в списке l только те элементы, котрые есть в списке pattern.
32.

```
func (l *list) InsertInOrder1(x int)
type data int
```

Исходный список упорядочен в неубывающем порядке. Вставить x в список так, чтобы порядок не нарушился.
33.

```
func (l *list) InsertInOrder1(x int)
type data int
```

Исходный список упорядочен по возрастанию и не содержит повторений. Если числа x нет в списке, то вставить его в список, не нарушая порядка; если же число x есть в списке, то не изменять список.
32.

```
func (l *list) Sort()
type data int
```

Отсортировать список в порядке неубывания.