

Линейные списки: стеки, очереди, деки

Лекция 3

Линейный список

- это множество, состоящее из n ($n \geq 0$) узлов (элементов) $X[1], X[2], \dots, X[n]$, структурные свойства которого ограничены линейным (одномерным) относительным положением узлов (элементов), т.е. следующими условиями:
 - если $n > 0$, то $X[1]$ – первый узел;
 - если $1 < k < n$,
то k -му узлу $X[k]$ предшествует узел $X[k-1]$,
а за узлом $X[k]$ следует узел $X[k+1]$;
 - $X[n]$ – последний узел.

Операции над линейными списками

1. Получить доступ к k-му элементу списка, проанализировать и/или изменить значения его полей.
2. Включить новый узел перед k-м.
3. Исключить k-й узел.
4. Объединить два или более линейных списков в один.
5. Разбить линейный список на два или более линейных списков.
6. Сделать копию линейного списка.
7. Определить количество узлов.
8. Выполнить сортировку в возрастающем порядке по некоторым значениям полей в узлах.
9. Найти в списке узел с заданным значением в некотором поле.
10. ... и т.д.

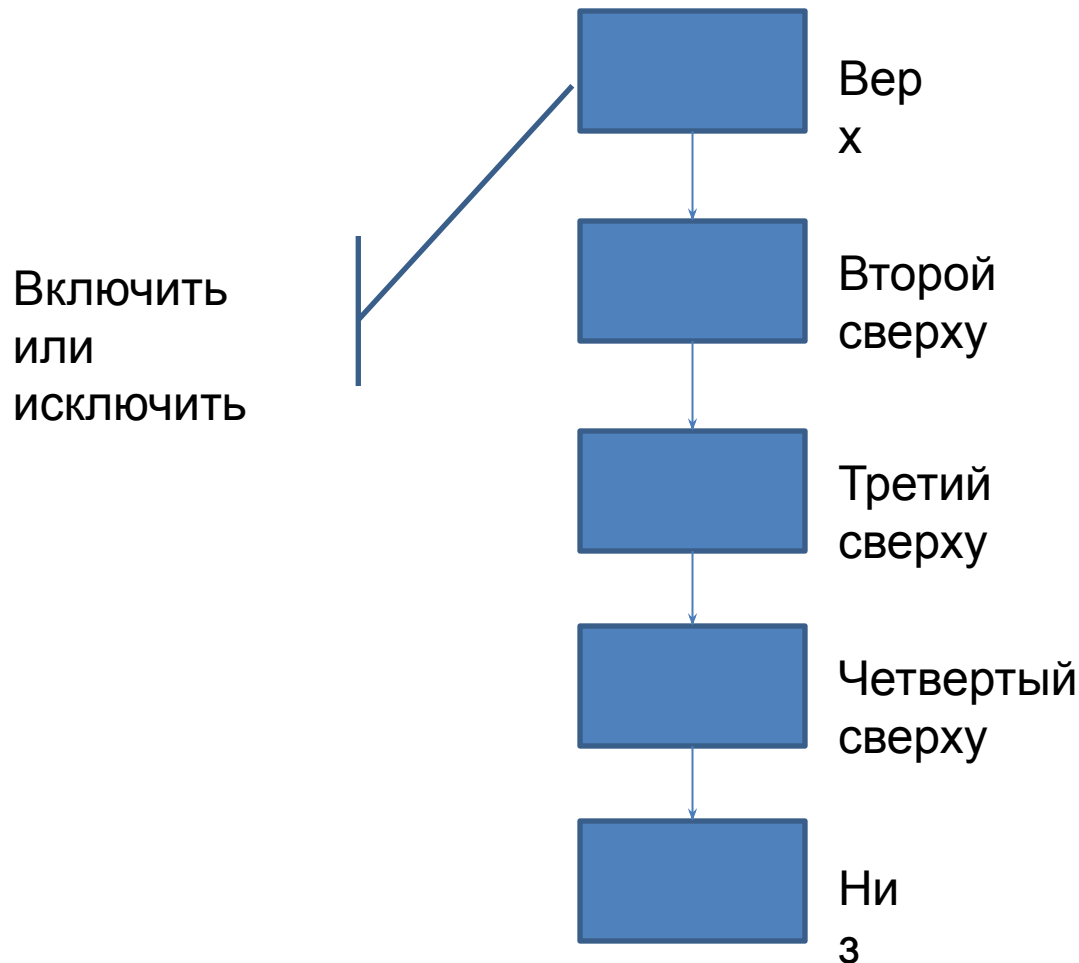
Не все операции нужны
одновременно!

=>

Будем различать типы линейных списков
по набору главных операций, которые
над ними выполняются.

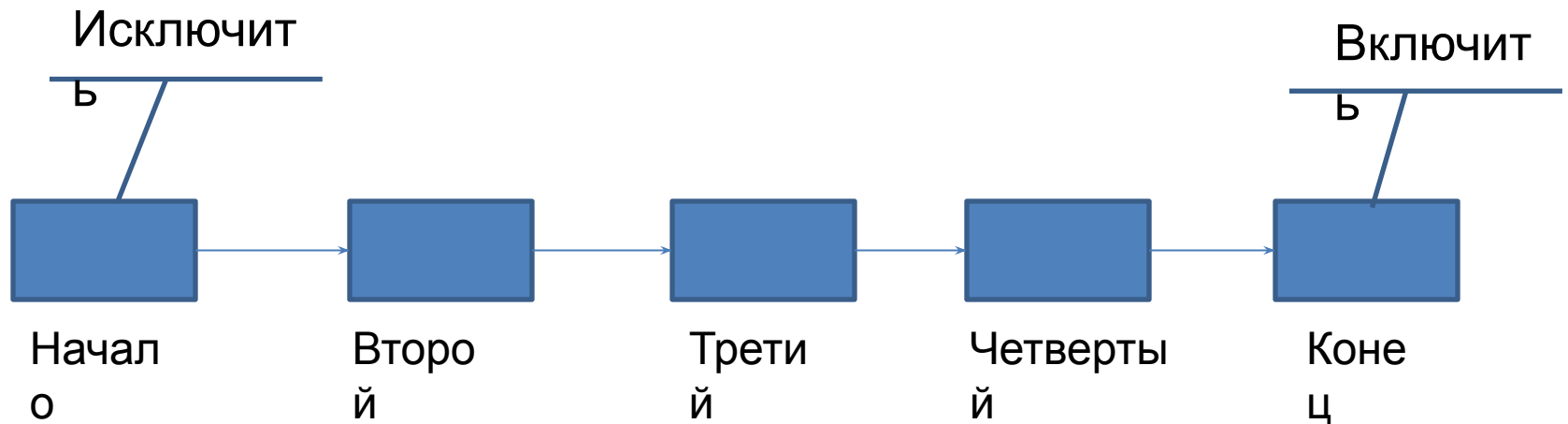
Стек

- это линейный список, в котором все включения и исключения (и всякий доступ) делаются в одном конце списка



Очередь

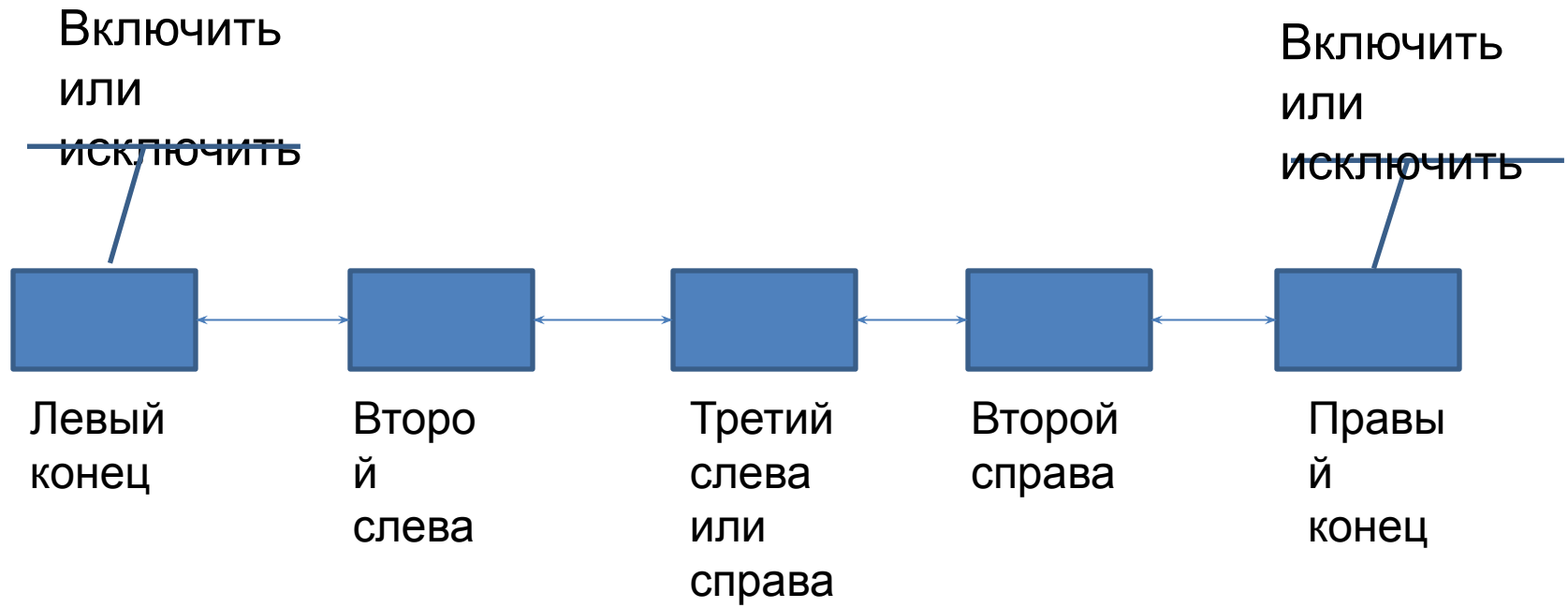
- это линейный список, в котором все включения производятся на одном конце списка, все исключения – на другом его конце.



Дек (double-ended queue)

очередь с двумя концами

- это линейный список, в котором все включения и исключения производятся на обоих концах списка



Стеки

- push-down список
- реверсивная память
- гнездовая память
- магазин
- LIFO (last-in-first-out)
- список йо-йо

Операции работы со стеками

1. `makenull (S)` – делает стек `S` пустым
2. `create(S)` – создает стек
3. `top (S)` – выдает значение верхнего элемента стека, не удаляя его
4. `pop(S)` – выдает значение верхнего элемента стека и удаляет его из стека
5. `push(x, S)` – помещает в стек `S` новый элемент со значением `x`
6. `empty (S)` - если стек пуст, то функция возвращает 1 (истина), иначе – 0 (ложь).

Реализация стека на си

```
struct list {  
    int data;  
    struct list * next;  
}  
  
typedef struct stack { struct list *top; } Stack;  
  
void makenull (Stack *S)  
{  
    struct list *p;  
    while (S->top)  
    {  
        p = S->top;  
        S->top = p->next;  
        free(p);  
    }  
}
```

Реализация стека на си - продолжение

```
Stack *create ()
{
    Stack *S;
    S = (Stack *)malloc(sizeof(Stack));
    S->top = NULL;
    return S;
}

int top (Stack *S)
{
    if (S->top)
        return (S->top->data);
    else
        return 0; //здесь может быть реакция на
        //ошибку – обращение к пустому стеку
}
```

Реализация стека на си -

продолжение

```
int pop(Stack *S)
{
    int a;
    struct list *p;
    p = S->top;
    a = p->data;
    S-> top = p->next;
    free(p);
    return a;
}
```

Реализация стека на си - продолжение

```
void push(int a, Stack *S)
{
    struct list *p;
    p = (struct list *) malloc ( sizeof (struct list));
    p->data = a;
    p->next = S-> top;
    S->top = p ;
}

int empty (Stack *S)
{
    return (S->top == NULL);
}
```

Виды записи выражений

- **Префиксная** (операция перед операндами)
- **Инфиксная** или скобочная (операция между операндами)
- **Постфиксная** или обратная польская (операция после операндов)

Примеры:

$a + (f - b * c / (z - x) + y) / (a * r - k)$ - инфиксная

$+a / + - f /* b c - z x y -* a r k$ - префиксная

$a f b c * z x - / - y + a r * k - / +$ -
постфиксная

Перевод из инфиксной формы в постфиксную

Вход: строка, содержащая арифметическое выражение, записанное в инфиксной форме

Выход: строка, содержащая то же выражение, записанное в постфиксной форме (обратной польской записи).

Обозначения:

числа, строки (идентификаторы) – операнды;

Знаки операций	Приоритеты операций
(	1
)	2
=	3
+, -	4
*, /	5

Алгоритм

Шаг 0:

Взять первый элемент из входной строки и поместить его в X.
Выходная строка и стек пусты.

Шаг 1:

Если X – операнд, то дописать его в конец выходной строки.

Если $X = '('$, то поместить его в стек.

Если $X = ')'$, то вытолкнуть из стека и поместить в конец выходной строки все элементы до первой встреченной открывающей скобки. Эту скобку вытолкнуть из стека.

Если X – знак операции, отличный от скобок, то пока стек не пуст, и верхний элемент стека имеет приоритет, больший либо равный приоритету X, вытолкнуть его из стека и поместить в выходную строку. Затем поместить X в стек.

Шаг 2:

Если входная строка не исчерпана, то поместить в X очередной элемент входной строки и перейти на Шаг 1, иначе пока стек не пуст, вытолкнуть из стека содержимое в выходную строку.

Перевод из инфиксной формы в постфиксную.

Пример

Входная строка:

a + (f - b * c / (z - x) + y) // (a * r - k)



X =

Выходная

строка:

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

Стек

:

Вычисления на стеке

Вход: строка, содержащая выражение, записанное в постфиксной форме.

Выход: число - значение заданного выражения.

Алгоритм:

Шаг 0:

Стек пуст.

Взять первый элемент из входной строки и поместить его в X.

Шаг 1:

Если X – операнд, то поместить его в стек.

Если X – знак операции, то вытолкнуть из стека два верхних элемента, применить к ним соответствующую операцию, результат положить в стек.

Шаг 2:

Если входная строка не исчерпана, то поместить в X очередной элемент входной строки и перейти на Шаг 1, иначе вытолкнуть из стека результат вычисления выражения.

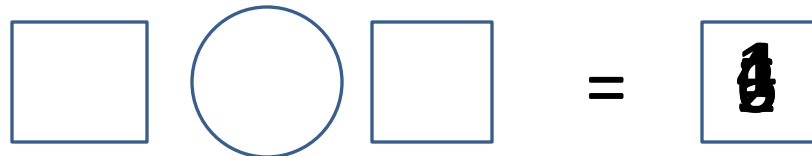
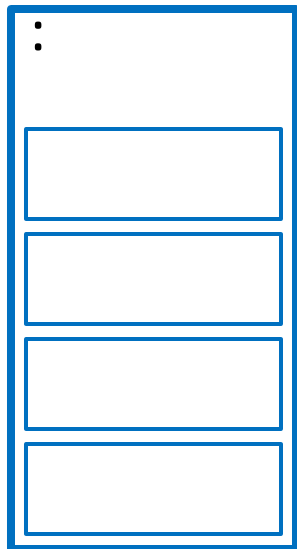
Вычисления на стеке. Пример

Входная строка:

5 2 3 * 4 2 // - 4 // + 1 -



Стек



Очереди

- FIFO (first-in-first-out) –первый вошел, первый вышел

Операции работы с очередями

1. `makenull (Q)` – делает очередь Q пустой
2. `create(Q)` – создает очередь
3. `first (Q)` – выдает значение первого элемента очереди, не удаляя его
4. `outqueue(Q)` – выдает значение первого элемента очереди и удаляет его из очереди
5. `inqueue(x, Q)` – помещает в конец очереди Q НОВЫЙ элемент со значением x
6. `empty (Q)` - если очередь пуста, то функция возвращает 1 (истина), иначе – 0 (ложь).

Реализация очереди на си

```
struct list
{
    int data;
    struct list * next;
}
typedef struct queue
{
    struct list *first;
    struct list *end;
} Queue;
```