

ПРОГРАММИРОВАНИЕ

Семинар 10.

Стеки, очереди, деки

Новосибирский государственный университет, 2019

Списки

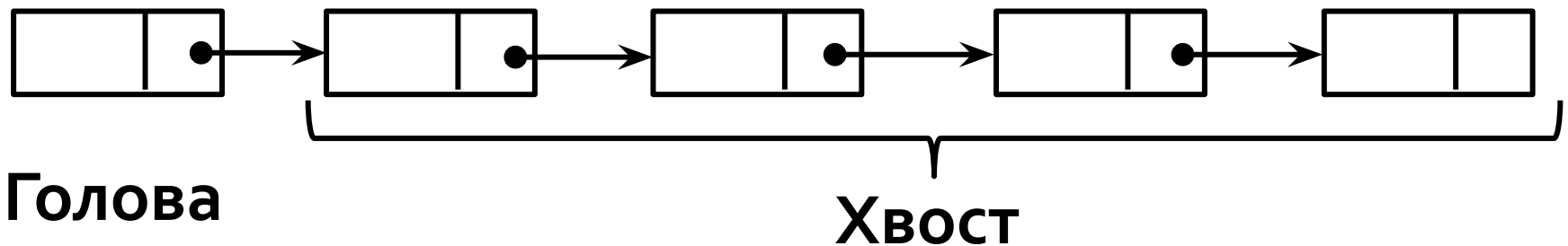
Список — структура данных, представляющая собой конечную последовательность связанных элементов.

Элемент списка:



Односвязные списки

Односвязный список — список, у элементов которого существует связь, указывающая на следующий элемент (односторонняя связь).



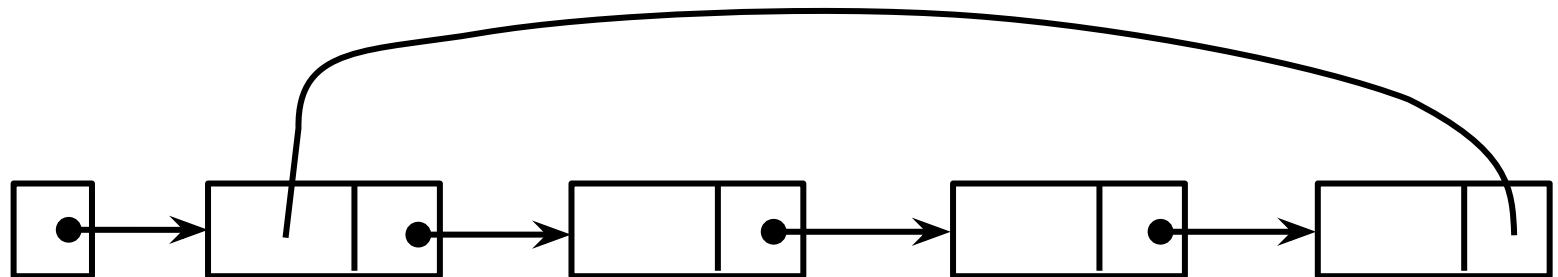
Односвязные списки

```
struct list {  
    int data;  
    struct list *next;  
};
```

```
struct list *head = NULL, *p, *t;
```

Циклические списки

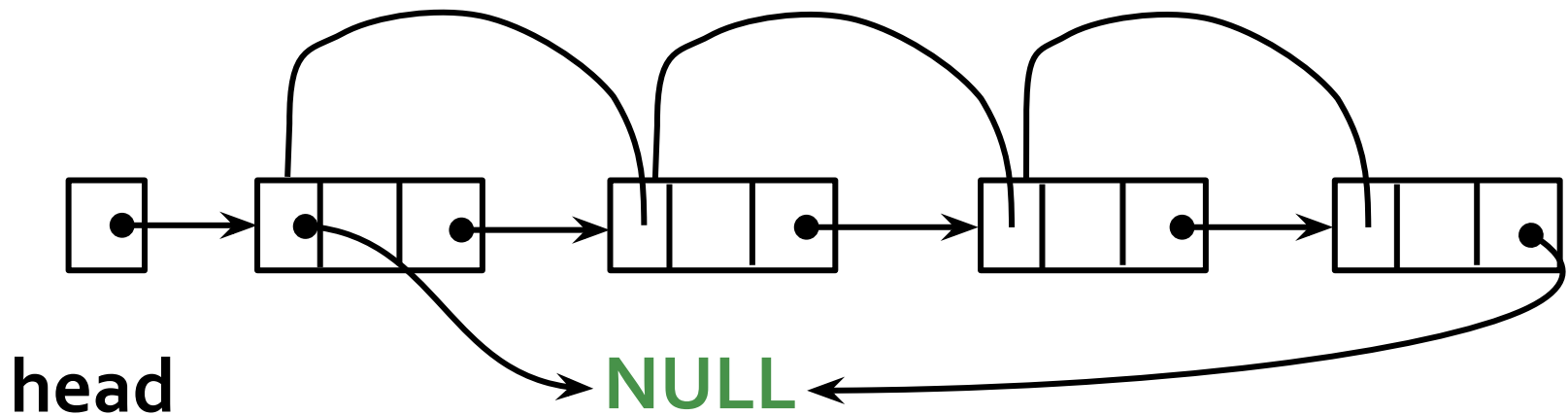
Циклический список — список, в котором связь указывает на первый или один из других элементов списка.



head

Двусвязные списки

Двусвязный список — список, элементы которого имеют по две связи, указывающие на предыдущий и следующий элементы.

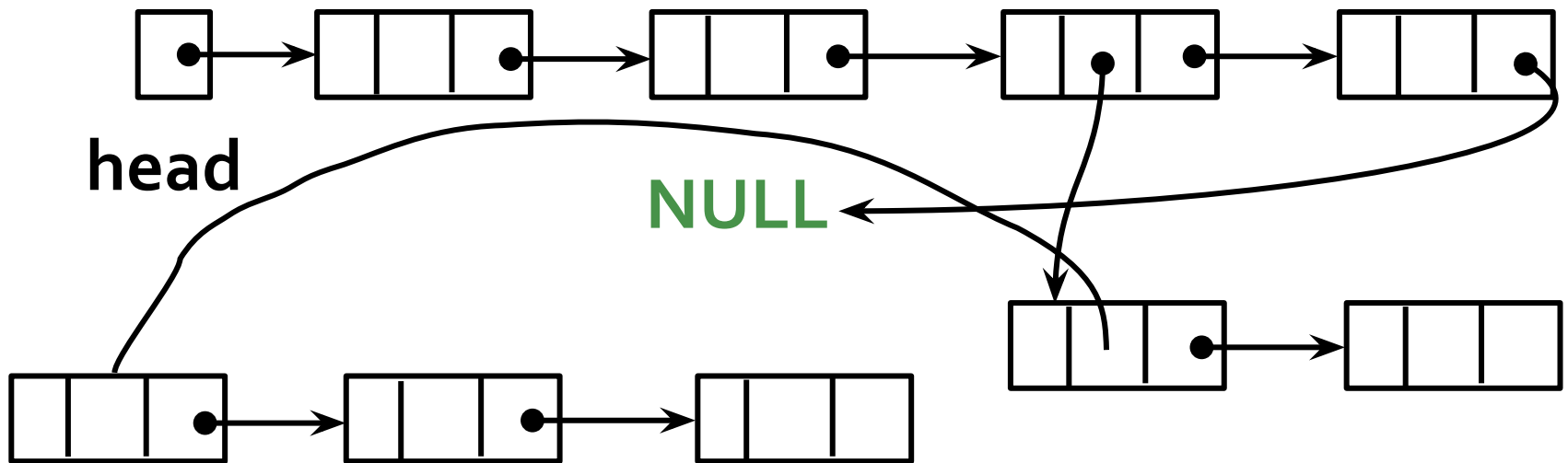


Двусвязные списки

```
struct List {  
    int data;  
    struct list *prev;  
    struct list *next;  
};
```

Иерархические списки

Иерархический список — список, значениями элементов которого являются указатели на другие списки (подсписки).



Операции над линейными списками

1. Получить доступ к некоторому элементу списка, проанализировать и / или изменить значения его полей.
2. Вставить новый элемент перед заданным.
3. Удалить заданный элемент.
4. Объединить два или более списков в один.
5. Разделить список на два или более.
6. Сделать копию списка.
7. Определить количество элементов.
8. Выполнить сортировку элементов по значениям определённых полей.
9. Найти элемент с заданным значением в некотором поле.

...и другие

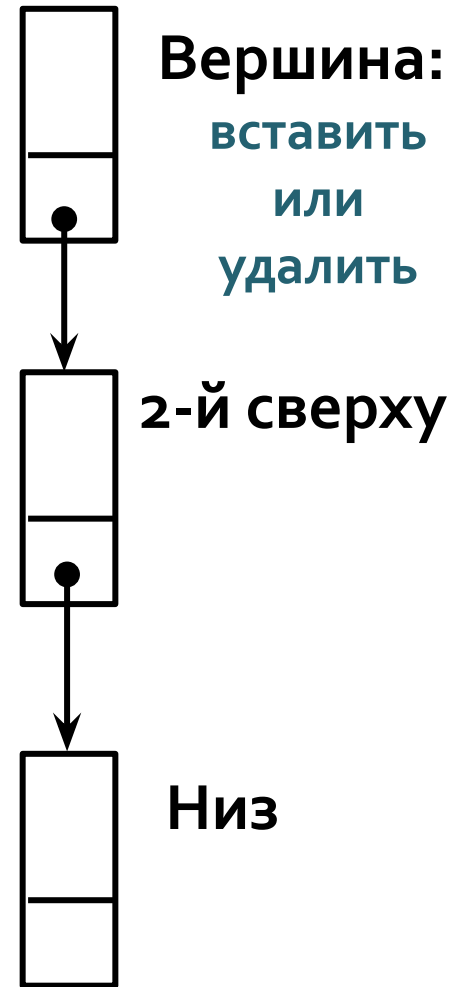
Операции над линейными списками

**Не все операции нужны
одновременно.**

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

Стек

Стек (*от англ. stack — штабель, стопка*) — линейный список, в котором все вставки, удаления и любой доступ производятся в одном конце списка.



Очередь

Очередь (*англ. queue*) — это линейный список, в котором все вставки производятся на одном конце, а все удаления — на другом.



Начало:
удалить

2-й

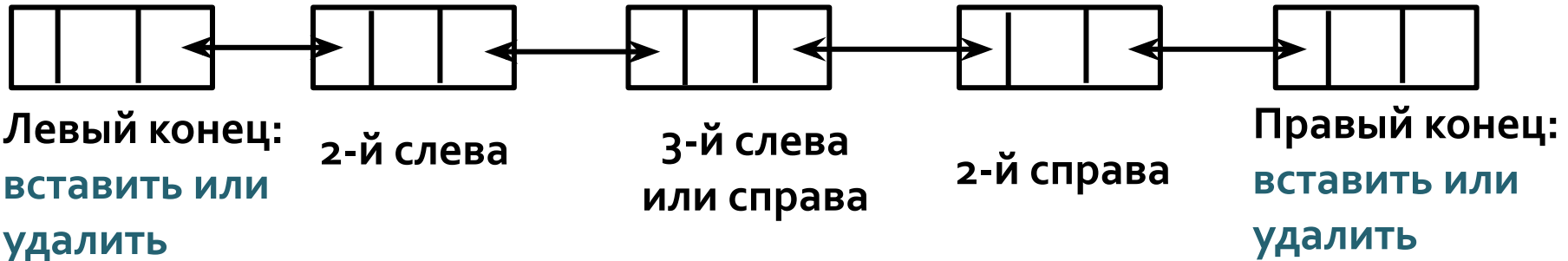
3-й

4-й

Конец:
вставить

Дек

Дек (*от англ. deque — double-ended queue — двусторонняя очередь*) — это линейный список, в котором все вставки и удаления производятся на обоих концах.



Стеки

Push-down список

Реверсивная память

Гнездовая память

Магазин

LIFO: last in first out

Список йо-йо

Операции со стеками

1. Опустошение.
2. Создание.
3. Получение значения вершины без удаления.
4. Получение значения вершины с удалением.
5. Добавление нового элемента.
6. Проверка пустоты.

Реализация стека на Си

```
struct list {  
    int data;  
    struct list *next;  
};  
typedef struct stack {struct list *top;} Stack;
```


Опустошение стека

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

Создание стека

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

Получение значения вершины без удаления

```
int top(Stack *S)
{
    if (S -> top)
        return (S -> top -> data);
    else
        return 0; /* или иная реакция на ошибку */
}
```

Получение значения вершины с удалением

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

Добавление нового элемента

```
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;
}
```

Проверка пустоты

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

Виды записи выражений

Префиксная: операция перед операндами.

Инфиксная (скобочная): операция между операндами.

Постфиксная (обратная польская): операция после операндов.

Виды записи выражений

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

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

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

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

Перевод из инфиксной формы в постфиксную

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.

Пример

Входная строка:

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



X =

Стек:

Выходная строка:

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

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.

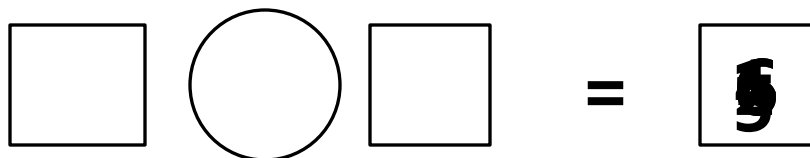
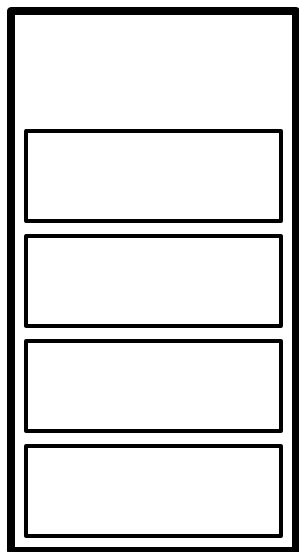
Пример

Входная строка:

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



Стек:



Очереди

FIFO: first in first out

Операции с очередями

1. Опустошение.
2. Создание.
3. Получение значения начала без удаления.
4. Получение значения начала с удалением.
5. Добавление нового элемента.
6. Проверка пустоты.

Реализация очереди на Си

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


Опустошение очереди

```
void makenull(Queue *Q)
{
    struct list *p;
    while (Q -> first) {
        p = Q -> first;
        Q -> first = p -> next;
        free(p);
    }
}
```

Создание очереди

```
Queue *create()
{
    Queue *Q;
    Q = (Queue *) malloc(sizeof(Queue));
    Q -> first = Q -> last = NULL;
    return Q;
}
```

Получение значения начала без удаления

```
int first(Queue *Q)
{
    if (Q -> first)
        return (Q -> first -> data);
    else
        return 0; /* или иная реакция на ошибку */
}
```

Получение значения начала с удалением

```
int outqueue(Queue *Q)
{
    int a;
    struct list *p;
    p = Q -> first;
    a = p -> data;
    Q -> first = p -> next;
    free(p);
    return a;
}
```

Добавление нового элемента

```
void inqueue(int a, Queue *Q)
{
    struct list *p;
    p = (struct list *) malloc(sizeof(struct list));
    p -> data = a;
    Q -> last -> next = p;
    p -> next = NULL;
    Q -> last = p;
}
```

Проверка пустоты

```
int empty(Queue *Q)
{
    return (Q -> first == NULL);
}
```