

Списки и другие абстрактные типы данных

лекция 8

План лекции

- Абстрактные типы данных
- АД список
 - Вставка и удаление элемента в список
- АД на основе списков
- Стек и примеры использования стеков
 - Перевод арифметического выражения из инфиксной в постфиксную запись
 - Вычисление значения выражения на стеке

Абстрактные типы данных



- Барбара Лисков р. 1939
- Стивен Жиль р. ?
- Liskov B., Zilles S. Programming with abstract data types // SIGPlan Notices, vol. 9, no. 4, 1974
 - Использование метода абстракции в программировании на примере построения польской записи выражения с помощью стека

Абстрактные типы данных

- Абстрактный тип данных – это набор операций над значениями этого АТД
 - Обязательно наличие операций для создания значений
- Реализация АТД – это отображение
 - Значение → содержимое памяти
 - Обычно не всё содержимое памяти
 - Операция --> послед-ть инструкций

АТД целое число

- Целые числа
 - Набор операций = { 0, 1, +, -, * }
 - Константы считаем 0-местными операциями
- Реализация на языке Си
 - Целое число → машинное представление
int
 - 0, 1, +, -, * --> машинные 0, 1, +, -, *

АТД список

- Создать пустой список
- Получить первую ячейку в списке
- Получить левую/правую соседку данной ячейки
- Создать новую ячейку списка перед/после данной
- Удалить ячейку списка перед/после данной
- Изменить/прочитать значение в данной ячейке списка
- Проверить наличие ячеек в списке

- Конечная последовательность ячеек, хранящих какие-то значения
- Адреса соседних ячеек списка в памяти могут отличаться больше, чем на размер ячейки

Классы реализаций списков

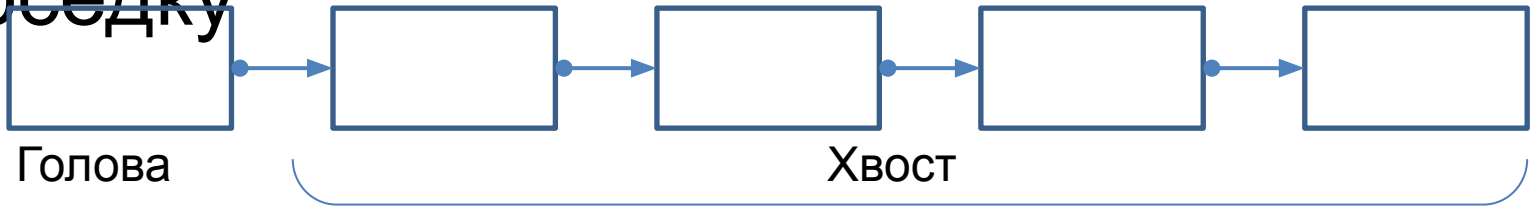
- Число соседак у ячейки -- 1 или 2
- Топология – линия или с циклом
- Тип значений – список или не список

- Один АДД может допускать несколько принципиально разных реализаций

Одно- и двусвязные списки

- Односвязный список – это список, каждая ячейка которого имеет ≤ 1

соседку



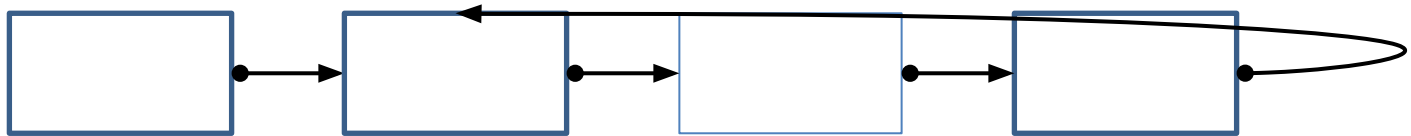
- Двусвязный список – это список, каждая внутренняя ячейка которого имеет две

соседки



Циклические списки

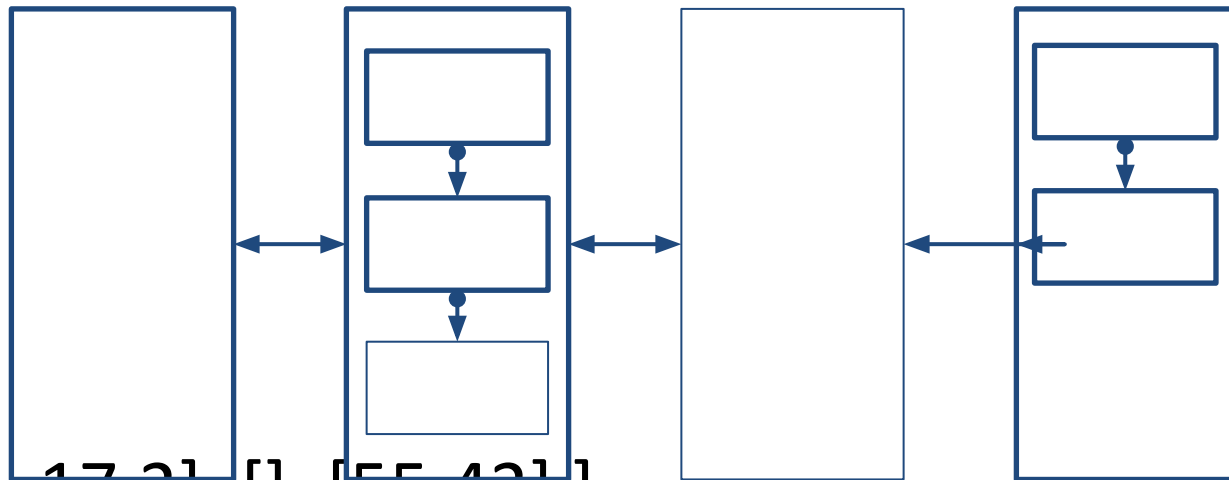
- Циклический список – это список, по ячейкам которого можно сколь угодно долго двигаться в одну из сторон



- Как определить, является ли список циклическим, не изменяя список и не используя дополнительной памяти?
- Почему рассматриваемый АД список не позволяет создавать циклические списки?
- Как сделать возможным создание циклических списков средствами АД список?

Иерархические списки

- Иерархический список -- это список, в ячейках которого хранятся списки
 - Списки могут быть разных классов



- [[], [5, -17, 2], [], [55, 42]]

Пример АДД список

`T` – тип элементов списка
`list_t` – список элементов типа `T`
`place_t` -- ячейка списка

```
list_t create(void);  
void insert_after(list_t *A, place_t p, T v);  
void erase_after(list_t *A, place_t p);  
void set_value(place_t p, T v);  
T get_value(place_t p);  
place_t prev(place_t p);  
place_t next(place_t p);  
place_t begin(list_t A);  
place_t end(void);
```

Пример использования АД СПИСОК

```
// Найти значение в списке
place_t find(list_t A, T v) {
    place_t p = begin(A);
    while (p != end()) {
        if (get_value(p) == v)
            return p;
        p = next(p);
    }
    return end();
}
// Перепишите с помощью for
```

Реализация 1 – типы

```
struct place_t {  
    T value;  
    struct place_t *next;  
};
```



```
struct list_t {  
    struct place_t *front;  
};
```

```
typedef struct list_t list_t;  
typedef struct place_t *place_t;
```

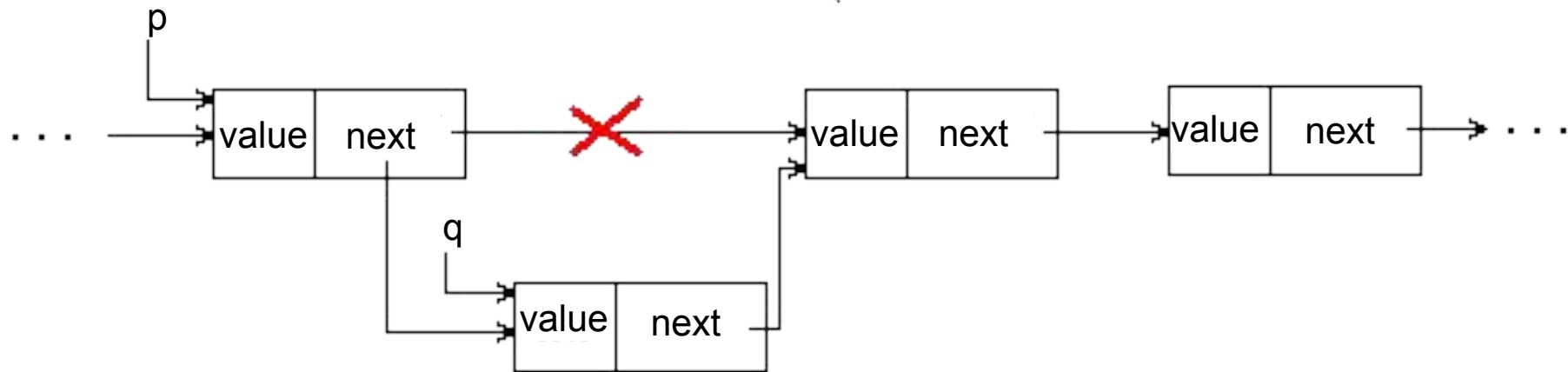
// К какому классу списков подходит такая реализация?

Реализация 1 – вставка ячейки

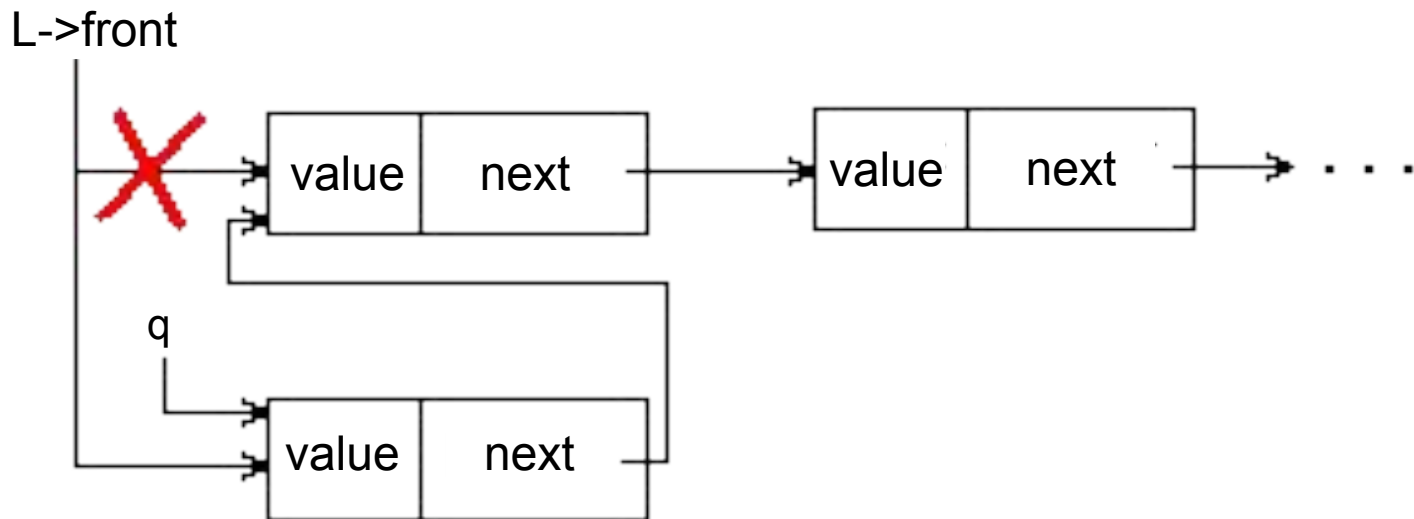
```
void insert_after(list_t *A, place_t p, T v) {  
    place_t q = malloc(sizeof *q); // q != NULL  
    q->value = v;  
    if (p == end()) // добавить первую ячейку  
        q->next = A->front, A->front = q;  
    else  
        q->next = p->next, p->next = q;  
}
```

```
// Напишите функцию  
// void insert(list_t *A, place_t p, T v);  
// добавляющую ячейку перед ячейкой p
```

Реализация 1 – вставка ячейки



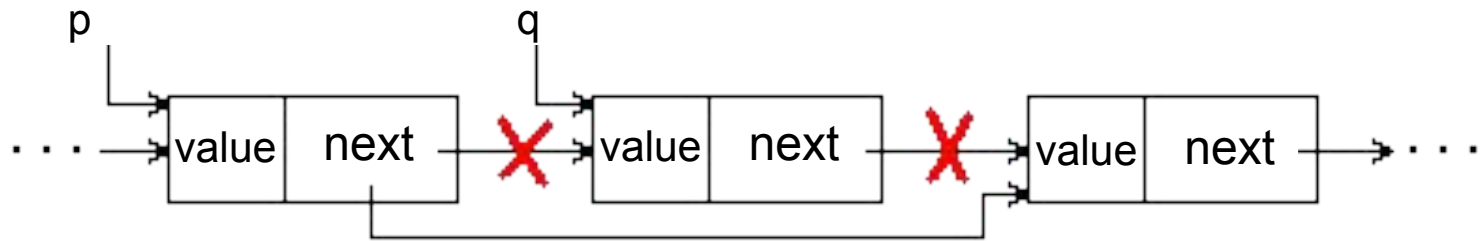
Реализация 1 – вставка ячейки



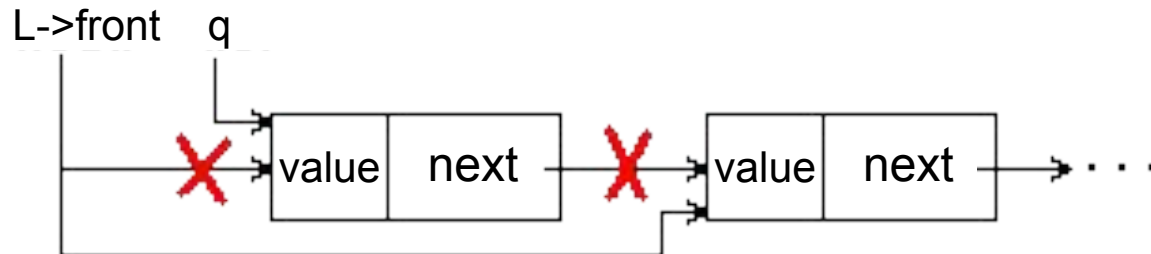
Реализация 1 – удаление ячейки

```
void erase_after(list_t *A, place_t p) {
    place_t *ptrp = p == end() ? &A->front
        : &p->next;
    if (*ptrp == end()) // удалять нечего
        return;
    place_t q = (*ptrp)->next;
    free(*ptrp);
    *ptrp = q;
}
// Напишите функцию
// void erase(list_t *A, place_t p);
// удаляющую ячейку p, а не next(p)
```

Реализация 1 – удаление ячейки



Из середины списка



Из начала списка

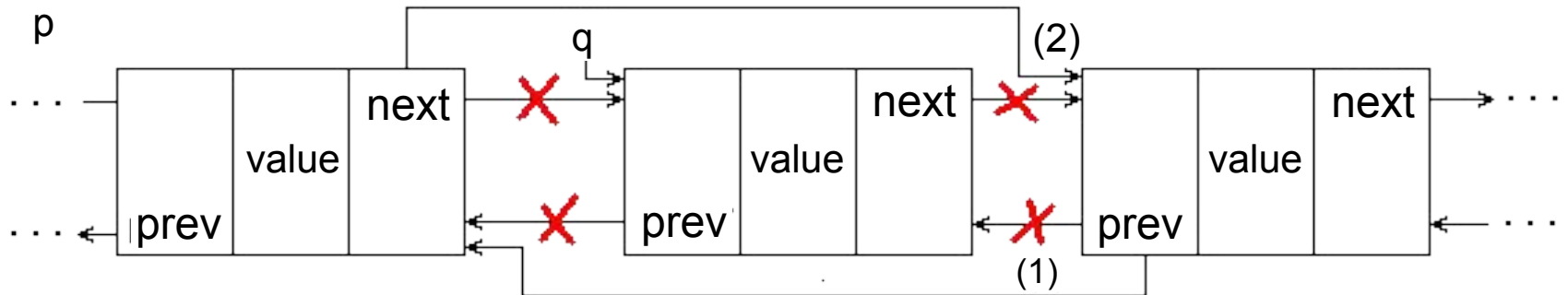
Реализация 2 – типы

```
struct place2_t {
    T          value;
    struct place2_t *   next, prev;
};
struct list2_t {
    struct place2_t *   front;
};
typedef struct list2_t   list2_t;
typedef struct place2_t * place2_t;

// К какому классу списков подходит
// такая реализация?
```

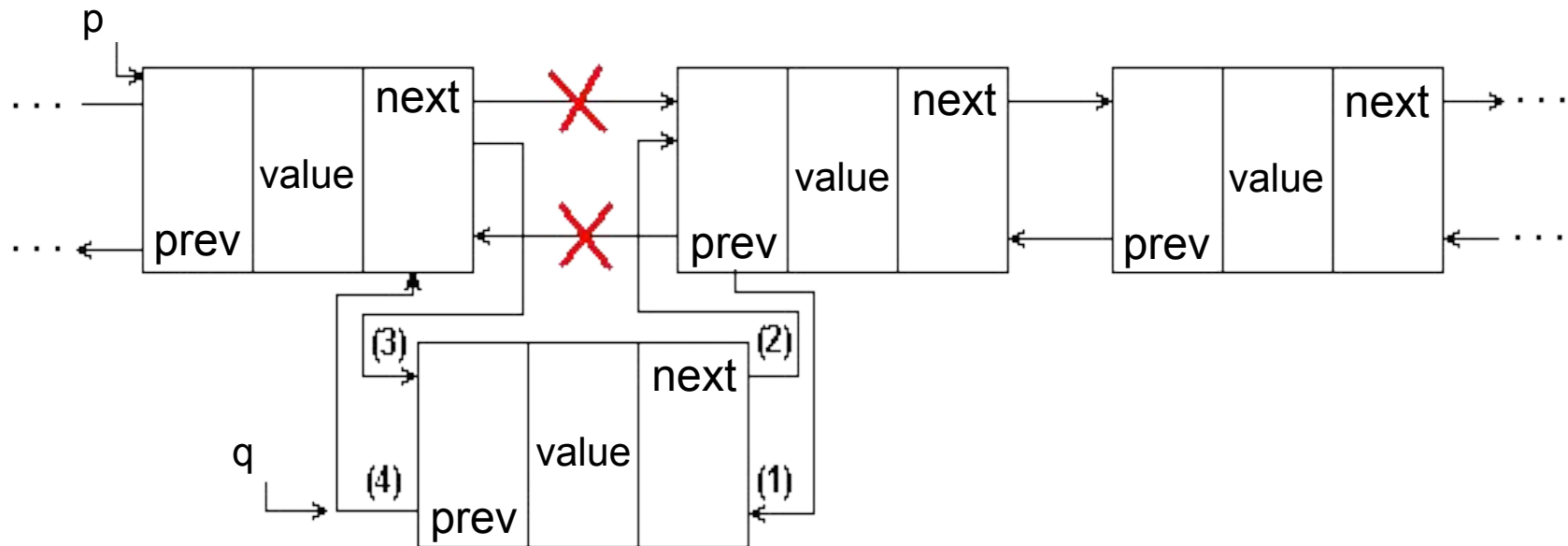
Реализация 2 – удаление ячейки

```
place2_t q = p->next;  
p->next->next->prev = p;    // (1)  
p->next = q -> next;      // (2)  
free(q);
```



Реализация 2 – вставка ячейки

```
place2_t q = malloc(sizeof *q); // p != NULL
p->next->prev = q; // (1)
q->next = p->next; // (2)
p->next = q; // (3)
q->prev = p; // (4)
```



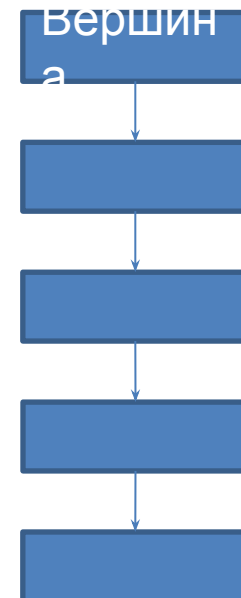
АТД на основе списков

- Стек (stack)
- Очередь (queue)
- Дек (double-ended queue)

- Сокращение набора операций
- Переиспользование готовой реализации
 - Увеличение производительности труда программиста

АТД стек

- Стек -- это список, в котором добавление/удаление ячеек происходит только на одном конце
- Последняя добавленная в стек ячейка называется вершиной стека
- реверсивная память
- гнездовая память
- магазин
- push-down список
- LIFO (last-in-first-out)
- список йо-йо



Операции работы со стеком

Обозначение	Смысл
create(S)	создать пустой стек
top(S)	вернуть значение на вершине
pop(S)	вернуть значение на вершине и удалить её
push(S, x)	добавить новую ячейку со значением x
empty(S)	проверить наличие ячеек в стеке
destroy(S)	уничтожить стек

Перевод из инфиксной записи в постфиксную запись

- Инфиксная или скобочная запись арифм. выражения
– $a + (f - b * c / (z - x) + y) / (a * r - k)$
- Префиксная запись
– $+a / + - f / * b c - z x y - * a r k$
- Постфиксная или обратная польская запись
– $a f b c * z x - / - y + a r * k - / +$
- Постфиксная запись = программа вычисления арифм. выражения
- Как из инфиксной записи получить постфиксную запись?

Перевод из инфиксной записи в постфиксную запись

- Вход: инфиксная запись арифметического выражения
- Выход: постфиксная запись того же арифметического выражения

Операция	Приоритет p
()	1
=	2
+ -	3
* /	4

Перевод из инфиксной записи в постфиксную запись

```
create(S), Выход = «»
пока Вход != «» повторять
    X = первый элемент Вход, удалить X из Вход
    если X – число, то Выход = Выход + X
    иначе если X = '(', то push(S, X)
    иначе если X = ')', то
        пока top(S) != '(' повторять
            Выход = Выход + pop(S)
        pop(S) // убрать самую '('
    иначе
        пока !empty(S) && p(top(S)) >= p(X) повторять
            Выход = Выход + pop(S)
        push(S, X)
пока !empty(S) повторять Выход = Выход + pop(S)
destroy(S)
```

Пример

Входная строка:

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



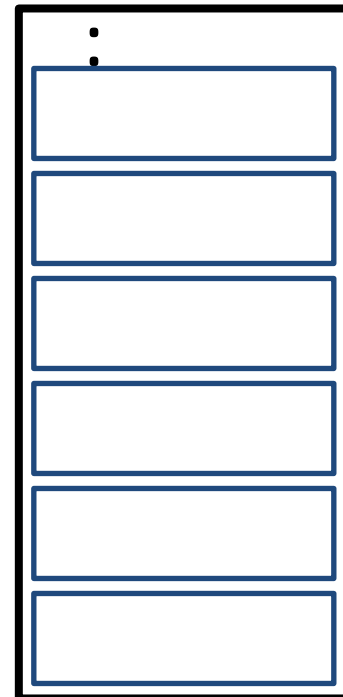
X =

Выходная

строка:

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

Стек



Вычисление арифметического выражения по постфиксной

записи

Вход: постфиксная запись выражение

Выход: значение выражения на входе

create(S)

пока Вход \neq «» повторять

 X = первый элемент Вход, удалить X из Вход

 Если X – число, то push(S, X)

 Если X – знак операции, то

 A=pop(S), B=pop(S), push(S, A X B)

Выход = pop(S)

destroy(S)

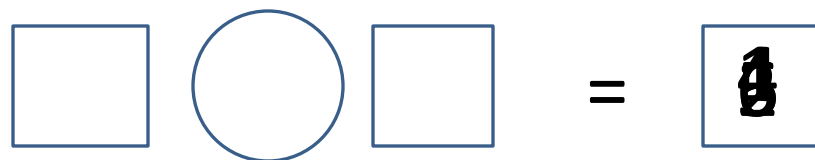
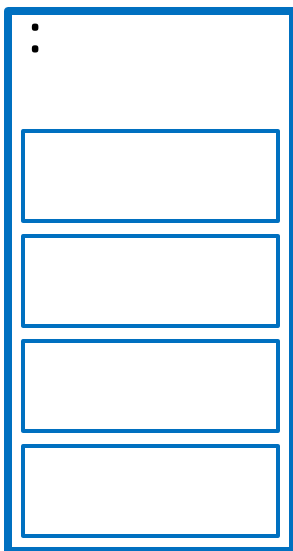
Пример

Входная строка:

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



Стек



Заключение

- Абстрактные типы данных
- Списки
 - Вставка и удаление элемента в список
- Стек и примеры использования стеков
 - Перевод арифметического выражения из инфиксной в постфиксную запись
 - Вычисление значения выражения на стеке