```
import math
import struct


def f11(x: int, y: int):
    return ((x**7 - y**5) / (x**7 - y**4 + 98)) \
        - math.sqrt(34 * x ** 8 + 50 * x ** 3) - (x ** 5 + 87 * x
** 3)


def f12(x: int):
    if x < 161:
        return math.sin(math.tan(x)) + x - 7
    elif 161 <= x < 259:
        return (x**7 / 42) - math.cos(x)
    elif 259 <= x < 302:
        return (76 * x**6) - math.log1p(x)
    elif x >= 302:
        return (29 * x**4) - (x / 33)


def f13(n: int, m: int):
    first, second = 0, 0
    for i in range(1, n + 1):
        for j in range(1, m + 1):
            first += math.sin(j) + math.tan(i) - 74
            second += j**2 - (i**8 / 74)
    return 35 * first - second / 36


def f14(n: int):
    return (f14(n - 1) / 43) + math.cos(f14(n - 1)) - 40 if n else 4


def f21(x: list):
    def check_0(first, second, third):
        if x[0] == 1990:
            return first
        elif x[0] == 1965:
            return second
        elif x[0] == 2000:
            return third

    def check_2(first, second):
        if x[2] == 1966:
            return first
        elif x[2] == 1958:
            return second

    if x[3] == 1984:
        return 9
    elif x[3] == 1972:
        if x[1] == 'ec':
            return check_0(0, check_2(1, 2), 3)
        elif x[1] == 'shell':
            return 4
```

```python
        elif x[1] == 'logos':
            return check_2(5, check_0(6, 7, 8))


def f22(x: int):
    C = (x & 0b00000100000000000000000000000000) << 5
    D = (x & 0b00111000000000000000000000000000) >> 1
    E = (x & 0b01000000000000000000000000000000)
    F = (x & 0b10000000000000000000000000000000) >> 2

    return (x & 0b00000011111111111111111111111111) | C | D | E | F


def f23(x: list) -> list:
    def delete_empty_cols(y: list) -> list:
        for key, value in enumerate(y):
            y[key] = list(filter(lambda l: l, value))
        return y

    def delete_multiple(y: list) -> list:
        n = []
        for index in y:
            n.append(index) if index not in n else None
        return n

    def break_first(row: list):
        first, second = row[0].split('#')
        row[0] = first.split()[-1]
        row.insert(1, second.split()[-1])

    def date_format(row: list, index: int):
        if row[index].count('-') == 2:
            row[index] = '/'.join(row[index].split('-')[::-1])

    def replace_at(row: list, index: int):
        if row[index].find('@') + 1:
            row[index] = row[index].replace('@', '[at]')

    x = delete_multiple(x)
    x = delete_empty_cols(x)

    for item in x:
        for i in range(len(item)):
            date_format(item, i)
            replace_at(item, i)
        break_first(item)

    return x


def f31(binary: bytes) -> dict:
    def struct_a(offset: int) -> dict:
        [a1] = struct.unpack('< d', binary[offset: offset + 8])
        offset += 8

        a2 = [struct_b(offset + i * (2 + 1 + 4 + 1)) for i in
range(6)]
```

```
        offset += 6 * (2 + 1 + 4 + 1)

        a3 = struct_c(offset)
        offset += 2 + 7

        [a4] = struct.unpack('< q', binary[offset: offset + 8])
        offset += 8

        a5 = struct_e(offset)
        offset += 4 + 4 + 1 + (7 * 4) + 2 + 3

        [a6, a7, a8] = \
            struct.unpack('< B l B', binary[offset: offset + 1 + 4 +
1])
        return {
            'A1': a1,
            'A2': a2,
            'A3': a3,
            'A4': a4,
            'A5': a5,
            'A6': a6,
            'A7': a7,
            'A8': a8,
        }

    def struct_b(offset: int) -> dict:
        [b1, b2, b3, b4] = \
            struct.unpack('< h b L B', binary[offset: offset + 2 + 1 +
4 + 1])
        return {
            'B1': b1,
            'B2': b2,
            'B3': b3,
            'B4': b4,
        }

    def struct_c(offset: int) -> dict:
        [c1] = struct.unpack('< H', binary[offset: offset + 2])
        offset += 2

        c2 = list(struct.unpack('< 7b', binary[offset: offset + 7]))
        return {
            'C1': struct_d(c1),
            'C2': c2,
        }

    def struct_d(offset: int) -> dict:
        [d1, d2, d3, d4, d5] = \
            struct.unpack('< b d l q d', binary[offset: offset + 1 + 8
+ 4 + 8 + 8])
        offset += 1 + 8 + 4 + 8 + 8

        d6 = list(struct.unpack('< 7B', binary[offset: offset + 7]))
        return {
            'D1': d1,
            'D2': d2,
            'D3': d3,
            'D4': d4,
            'D5': d5,
            'D6': d6,
        }

    def struct_e(offset: int) -> dict:
        [e1, e2, e3] = \
            struct.unpack('< L f B', binary[offset: offset + 4 + 4 +
1])
        offset += 4 + 4 + 1

        e4 = list(struct.unpack('< 7I', binary[offset: offset + (7 *
4)]))
        offset += 7 * 4

        [e5] = struct.unpack('< H', binary[offset: offset + 2])
        offset += 2

        e6 = list(struct.unpack('< 3b', binary[offset: offset + 3]))
        return {
            'E1': e1,
            'E2': e2,
            'E3': e3,
            'E4': e4,
            'E5': e5,
            'E6': e6,
        }

    return struct_a(5)


class C32:
    def __init__(self):
        self.state: C32.State = C32.A(self)

    def bolt(self) -> int:
        return self.state.bolt()

    def group(self) -> int:
        return self.state.group()

    def tail(self) -> int:
        return self.state.tail()

    class State:
        def __init__(self, parent):
            self.parent: C32 = parent

        def bolt(self) -> int:
            raise RuntimeError

        def group(self) -> int:
            raise RuntimeError

        def tail(self) -> int:
            raise RuntimeError
```

```
class A(State):
    def tail(self):
        self.parent.state = C32.B(self.parent)
        return 0

    def group(self):
        self.parent.state = C32.D(self.parent)
        return 1

class B(State):
    def bolt(self):
        self.parent.state = C32.C(self.parent)
        return 2

    def group(self):
        self.parent.state = C32.G(self.parent)
        return 3

    def tail(self):
        self.parent.state = C32.D(self.parent)
        return 4

class C(State):
    def group(self):
        self.parent.state = C32.D(self.parent)
        return 5

class D(State):
    def tail(self):
        self.parent.state = C32.E(self.parent)
        return 6

class E(State):
    def group(self):
        self.parent.state = C32.F(self.parent)
        return 7

class F(State):
    def tail(self):
        self.parent.state = C32.G(self.parent)
        return 8

class G(State):
    def bolt(self):
        self.parent.state = C32.E(self.parent)
        return 9
```