```python
"""
COSC364 2022-S1 Assignment: RIP routing
Authors: MENG ZHANG (71682325), ZHENG CHAO (21671773)
File: IO_parser.py
"""

def router_config(file_name):
    """
    Parameter:
    file_name: string
    file format:
    i.e.
    ------------------------------
    router-id 2
    input-ports 6020, 6021
    output-ports 6010-1-1, 6030-2-3
    period 3
    timeout 18
    ------------------------------

    Return: config_data
    a dictionary with 4 keys of router_id, input_ports, output_ports,
    timers
    i.e. {'router_id': 2, 'input_ports': [6020, 6021],
    'output_ports_metric_id': {6010: {'metric': 1, 'router_id': 1},
                               6020: {...}}, 'period': 3, 'timeout': 18}
    """
    raw_config = read_config(file_name)
    config_data = parse_config(raw_config)
    return config_data


def read_config(file_name):
    """
    Parameter:
    file_name: string.
    file format:
    i.e.
    ------------------------------
    router-id 2
    input-ports 6020, 6021
    output-ports 6010-1-1, 6030-2-3
    period 3
    timeout 18
    ------------------------------

    Return: a list of strings with 4 elements.
    i.e. ['router-id 2', 'input-ports 6020, 6021', 'output-ports 6010-1-1,
          6030-2-3', 'period 3', 'timeout 18']
    """
    try:
        with open(file_name) as config_file:
            raw_config = config_file.read().splitlines()
            return raw_config
    except FileNotFoundError:
        print("Error: the config file name is invalid")


def parse_config(raw_config):
    """
    Parameter:
    raw_config: a list of strings with 4 elements.
    i.e. ['router-id 2', 'input-ports 6020, 6021', 'output-ports 6010-1-1,
          6030-2-3', 'period 3', 'timeout 18']

    Return: config_data
    a dictionary with 4 keys of router_id, input_ports, output_ports,
    timers
    i.e. {'router_id': 2, 'input_ports': [6020, 6021],
          'output_ports_metric_id': {6010: {'metric': 1, 'router_id': 1},
                                     6020: {...}}, 'period': 3, 'timeout': 18}
    """
    try:
        # get router id
        router_id = parse_id(raw_config[0])
        # get input ports
        input_ports = parse_input_ports(raw_config[1])
        # check if input ports contains duplicate ports
        if contains_duplicates(input_ports):
            raise ValueError("The input ports contains duplicate ports")
        # get output ports
        output_ports, output_ports_metric_id = parse_output_ports(raw_config[2])
        # check if input ports and output ports contain duplicate ports
        if duplicate_lists(input_ports, output_ports):
            raise ValueError("The input ports and output ports contain duplicate ports")
        # get period
        period = parse_period(raw_config[3])
        # get timeout
        timeout = parse_timeout(raw_config[4])
        # check timeout vs period ratio
        if not is_valid_timer_ratio(period, timeout):
            raise ValueError("The ratio timeout vs period should be 6")
        # create coinfig_data dictionary
        config_data = {"router_id": router_id, "input_ports": input_ports,
                       "output_ports_metric_id": output_ports_metric_id,
                       "period": period, "timeout": timeout}
        return config_data
    except IndexError as ie:
        print(ie)
        print("Some value of the config file is not available")
    except ValueError as ve:
        print(ve)
        print("Some value of the config file is invalid")


def parse_id(raw_id):
    """
    Parameter:
    raw_id: a string
    i.e. 'router-id 2'

    Return: router_id
    an interger between 1 and 64000 i.e. 1
    """
    try:
        router_id = int(raw_id.split()[1])
        if (router_id < 1 or router_id > 64000):
            raise ValueError("Router ID value is out of bounds")
        return router_id
    except IndexError as e:
        print(e)
        print("The config router ID value is not available")
    except ValueError as e:
        print(e)
        print("The config router ID value must be an integer between 1 and 64000")


def parse_input_ports(raw_input_ports):
    """
    Parameter:
    raw_input_ports: a string
    i.e 'input-ports 6020, 6021'
```

```python

    Return: input_ports
    a list of integers which are between 1024 and 64000
    i.e. [6020, 6021]
    """
    try:
        input_ports_temp = raw_input_ports.split()[1:]
        input_ports = []
        for port_str in input_ports_temp:
            port_int = int(port_str.strip(','))
            if (port_int < 1024 or port_int > 64000):
                raise ValueError("Input port value is out of bounds")
            input_ports.append(port_int)
        return input_ports
    except IndexError as e:
        print(e)
        print("The config input port value is not available")
    except ValueError as e:
        print(e)
        print("The config input port value must be an integer between 1024 and 64000")


def parse_output_ports(raw_output_ports):
    """
    Parameter:
    raw_input_ports: a string
    i.e 'output-ports 6010-1-1, 6030-2-3'

    Return: output_ports, output_ports_metric_id
    output_ports: a list of integers which are between 1024 and 64000
    i.e. [6010, 6030]
    output_ports_metric_id: a dict of dicts in which key is port number
    and each sub dict contains key(port)'s metric and id.
    Metric > 0, 1 <= ID <= 64000
    i.e. {6010: {'metric': 1, 'router_id': 1}, 6020: {...}}
    """
    try:
        output_ports_combo_temp = raw_output_ports.split()[1:]
        output_ports = []
        output_ports_metric_id = {}
        for port_combo_str in output_ports_combo_temp:
            port_combo_temp = port_combo_str.strip(',').split('-')
            port_int = int(port_combo_temp[0])
            metric_int = int(port_combo_temp[1])
            id_int = int(port_combo_temp[2])
            if (port_int < 1024 or port_int > 64000):
                raise ValueError("Ouput port value is out of bounds")
            if metric_int < 1:
                raise ValueError("Output port metric is out of bounds")
            if id_int < 1 or id_int > 64000:
                raise ValueError("Output id is out of bounds")
            output_ports.append(port_int)
            # output_ports_metric_id.append([port_int, metric_int, id_int])
            output_ports_metric_id[port_int] = {'metric': metric_int,
                                                'router_id': id_int}
        return output_ports, output_ports_metric_id
    except IndexError as e:
        print(e)
        print("The config output port value is not available")
    except ValueError as e:
        print(e)
        print("The config output ports must be fomatted as port-metric-id")
        print("The config output port value must be an integer between 1024 and 64000")
        print("The config output port metric must be an integer greater than 0")
        print("The config output port id must be an integer between 1 and 64000")

```

```python

def parse_period(raw_period):
    """
    Parameter:
    raw_period: a string
    i.e. 'period 3'

    Return: period
    period: a positive integer
    i.e. 3
    """
    try:
        period = int(raw_period.split()[1])
        if period < 1:
            raise ValueError("Router period value is out of bounds")
        return period
    except IndexError as e:
        print(e)
        print("The config router period value is not available")
    except ValueError as e:
        print(e)
        print("The config router timeout value must be a positive integer")

def parse_timeout(raw_timeout):
    """
    Parameter:
    raw_timeout: a string
    i.e. 'timeout 18'

    Return: timeout
    timeout: a positive integer
    i.e. 18
    """
    try:
        timeout = int(raw_timeout.split()[1])
        if timeout < 1:
            raise ValueError("Router timeout value is out of bounds")
        return timeout
    except IndexError as e:
        print(e)
        print("The config router timeout value is not available")
    except ValueError as e:
        print(e)
        print("The config router timeout value must be a positive integer")


def contains_duplicates(lst):
    """
    Parameter:
    lst: a list

    Return: boolean
    if the lst contains duplicates, return true, otherwise false
    """
    return len(set(lst)) != len(lst)

def duplicate_lists(lst1, lst2):
    """
    Parameters:
    lst1: a list
    lst2: a list

    Return: boolean
    if the two lists contains duplicate items, return true, otherwise false
    """
    return len(set(lst1).union(set(lst2))) != len(lst1) + len(lst2)
```

```python

def is_valid_timer_ratio(period, timeout):
    """
    Parameters:
    period: a positive integer
    period: a positive integer

    Return: boolean
    if timeout / period = 6, return true, otherwise false
    """
    return timeout / period == 6
```