```
import functools
import operator
import os
from typing import List

from dace.types import typeclass

import helper
from base_node_class import BaseKernelNodeClass
from bounded_queue import BoundedQueue


class Output(BaseKernelNodeClass):
    """
        The Output class is a subclass of the BaseKernelNodeClass and represents an Ouput node in the KernelChainGraph.
        Its purpose is to store data coming from the pipeline/dataflow design.
    """

    def __init__(self,
                 name: str,
                 data_type: typeclass,
                 dimensions: List[int],
                 data_queue=None) -> None:
        """
        Initializes the Output class with given initialization parameters.
        :param name: name of the output node
        :param data_type: data type of the data feed into output
        :param dimensions: global problem dimensions
        :param data_queue: dummy
        """
        # init superclass with queue of size: global problem size
        super().__init__(name=name, data_type=data_type, data_queue=BoundedQueue(name="output",
                                                                                 maxsize=functools.reduce(operator.mul,
                                                                                                          dimensions),
                                                                                 collection=[]))

    def reset_old_compute_state(self) -> None:
        """
        Reset compute-specific internal state (only for Kernel node).
        """
        pass  # nothing to do

    def try_read(self) -> None:
        """
        Read data from predecessor.
        """
        # check for single input
        assert len(self.inputs) == 1  # there should be only a single one
        for inp in self.inputs:
```

```
        # read data
        if self.inputs[inp]["delay_buffer"].try_peek_last() is not False and self.inputs[inp]["delay_buffer"]\
                .try_peek_last() is not None:
            self.data_queue.enqueue(self.inputs[inp]["delay_buffer"].dequeue())
            self.program_counter += 1
        elif self.inputs[inp]["delay_buffer"].try_peek_last() is not False:
            self.inputs[inp]["delay_buffer"].dequeue()  # remove bubble

def try_write(self) -> None:
    """
    Feed data to all successor channels (for Input and Kernel nodes)
    """
    pass  # nothing to do

def write_result_to_file(self,
                         input_config_name: str) -> None:
    """
    Write internal queue with computation result to the file results/INPUT_CONFIG_NAME/SELF.NAME_simulation.dat
    :param input_config_name: the config name, used to determine the save path
    """
    # join the paths
    output_folder = os.path.join("results", input_config_name)
    # create (recursively) directories
    os.makedirs(output_folder, exist_ok=True)
    # store the data
    helper.save_array(self.data_queue.export_data(), "{}/{}_{}.dat".format(output_folder, self.name, 'simulation'))
```