```
import numpy as np
from dace.types import typeclass

from base_node_class import BaseKernelNodeClass
from bounded_queue import BoundedQueue


class Input(BaseKernelNodeClass):
    """
        The Input class is a subclass of the BaseKernelNodeClass and represents an Input node in the KernelChainGraph.
        Its purpose is to feed input array data into the pipeline/dataflow design.
    """

    def __init__(self,
                 name: str,
                 data_type: typeclass,
                 data_queue: BoundedQueue = None) -> None:
        """
        Initialize the Input node.
        :param name: node name
        :param data_type: data type of the data
        :param data_queue: BoundedQueue containing the input data
        """
        # initialize superclass
        super().__init__(name=name, data_queue=data_queue, data_type=data_type)
        # set internal fields
        self.queues = dict()
        self.dimension_size = data_queue.maxsize
        self.init = False  # flag for internal initialization (must be done later when all successors are added to the
        # graph)

    def init_queues(self):
        """
        Create individual queues for all successors in order to feed data individually into the channels.
        """
        # add a queue for each successor
        self.queues = dict()
        for successor in self.outputs:
            self.queues[successor] = BoundedQueue(name=successor,
                                                  maxsize=self.data_queue.maxsize,
                                                  collection=self.data_queue.export_data())
        self.init = True  # set init flag

    def reset_old_compute_state(self):
        """
        Reset compute-specific internal state (only for Kernel node).
        """
        pass  # nothing to do
```

```
def try_read(self):
    """
    Read data from predecessor (only for Kernel and Output node).
    """
    pass  # nothing to do

def try_write(self):
    """
    Feed data to all successor channels.
    :return:
    """
    # set up all individual data queues
    if not self.init:
        self.init_queues()
    # feed data into pipeline inputs (all kernels that feed from this input data array)
    for successor in self.outputs:
        if self.queues[successor].is_empty() and not self.outputs[successor]["delay_buffer"].is_full():  # no more
            # data to feed, add bubble
            self.outputs[successor]["delay_buffer"].enqueue(None)  # insert bubble
        elif self.outputs[successor]["delay_buffer"].is_full():  # channel full, skip
            pass
        else:  # feed data into channel
            data = self.queues[successor].dequeue()
            self.outputs[successor]["delay_buffer"].enqueue(data)
            self.program_counter = self.dimension_size - max([self.queues[x].size() for x in self.queues])

def init_input_data(self, inputs):
    """
    Initialize internal queue i.e. read data from config or external file.
    :param inputs:
    :return:
    """
    # check if data is in the config or in a separate file
    if isinstance(inputs[self.name]["data"], list):  # inline
        self.data_queue.import_data(inputs[self.name]["data"])
    elif isinstance(inputs[self.name]["data"], str):  # external file
        coll = None
        if inputs[self.name]["data"].lower().endswith(('.dat', '.bin', '.data')):  # general binary data file
            coll = np.fromfile(inputs[self.name]["data"], inputs[self.name]["data_type"].type)
        if inputs[self.name]["data"].lower().endswith('.h5'):  # h5 file
            from h5py import File
            f = File(inputs[self.name]["data"], 'r')
            coll = np.array(list(f[list(f.keys())[0]]), dtype=inputs[self.name]["data_type"].type)  # read data
            # from first key
        elif inputs[self.name]["data"].lower().endswith('.csv'):  # csv file
            coll = list(np.genfromtxt(inputs[self.name]["data"], delimiter=',',
                                      dtype=inputs[self.name]["data_type"].type))
        # add data to queue
        self.data_queue.import_data(coll)
```

```
else:
    raise Exception("Input data representation should either be implicit (list) or a path to a csv file.")
```