```
class _DoublyLinkedBase:
  """A base class providing a doubly linked list representation."""

  #---------------------------- nested _Node class ----------------------------
  # nested _Node class
  class _Node:
    """Lightweight, nonpublic class for storing a doubly linked node."""
    __slots__ = '_element', '_prev', '_next'            # streamline
memory

    def __init__(self, element, prev, next):            # initialize
node's fields
      self._element = element                           # user's element
      self._prev = prev                                 # previous node
reference
      self._next = next                                 # next node
reference

  #---------------------------- list constructor ----------------------------

  def __init__(self):
    """Create an empty list."""
    self._header = self._Node(None, None, None)
    self._trailer = self._Node(None, None, None)
    self._header._next = self._trailer                  # trailer is after
header
    self._trailer._prev = self._header                  # header is before
trailer
    self._size = 0                                      # number of
elements

  #---------------------------- public accessors ----------------------------

  def __len__(self):
    """Return the number of elements in the list."""
    return self._size

  def is_empty(self):
    """Return True if list is empty."""
    return self._size == 0

  #---------------------------- nonpublic utilities ----------------------------
--
```

```
  def _insert_between(self, e, predecessor, successor):
    """Add element e between two existing nodes and return new node."""
    newest = self._Node(e, predecessor, successor)      # linked to
neighbors
    predecessor._next = newest
    successor._prev = newest
    self._size += 1
    return newest

  def _delete_node(self, node):
    """Delete nonsentinel node from the list and return its element."""
    predecessor = node._prev
    successor = node._next
    predecessor._next = successor
    successor._prev = predecessor
    self._size -= 1
    element = node._element                             # record deleted
element
    node._prev = node._next = node._element = None      # deprecate node
    return element
```